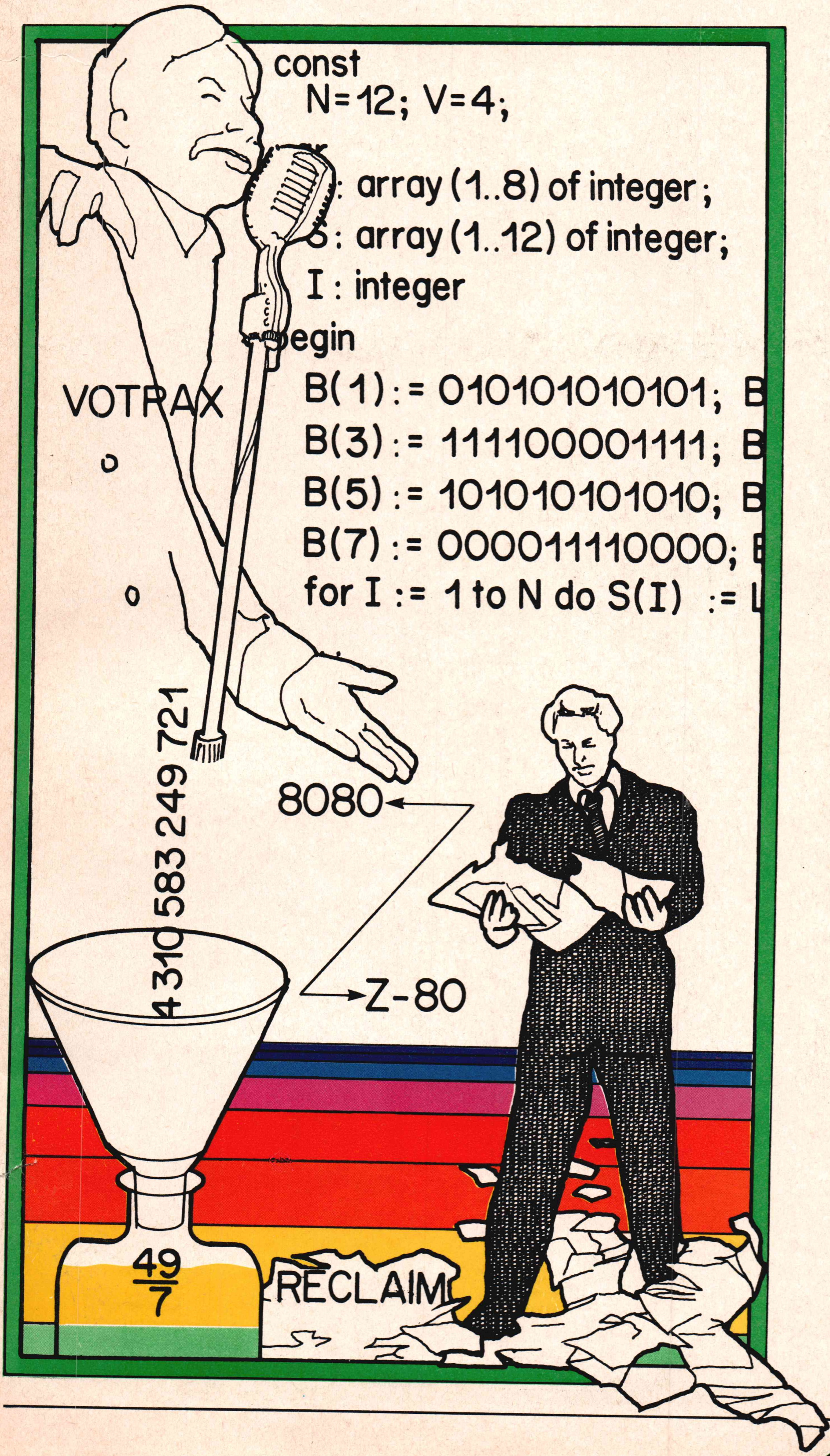
#78 April 1983 ($3.00 in Canada)

Dr. Dobb's Journal

For Users of Small Computer Systems



RECLAIM

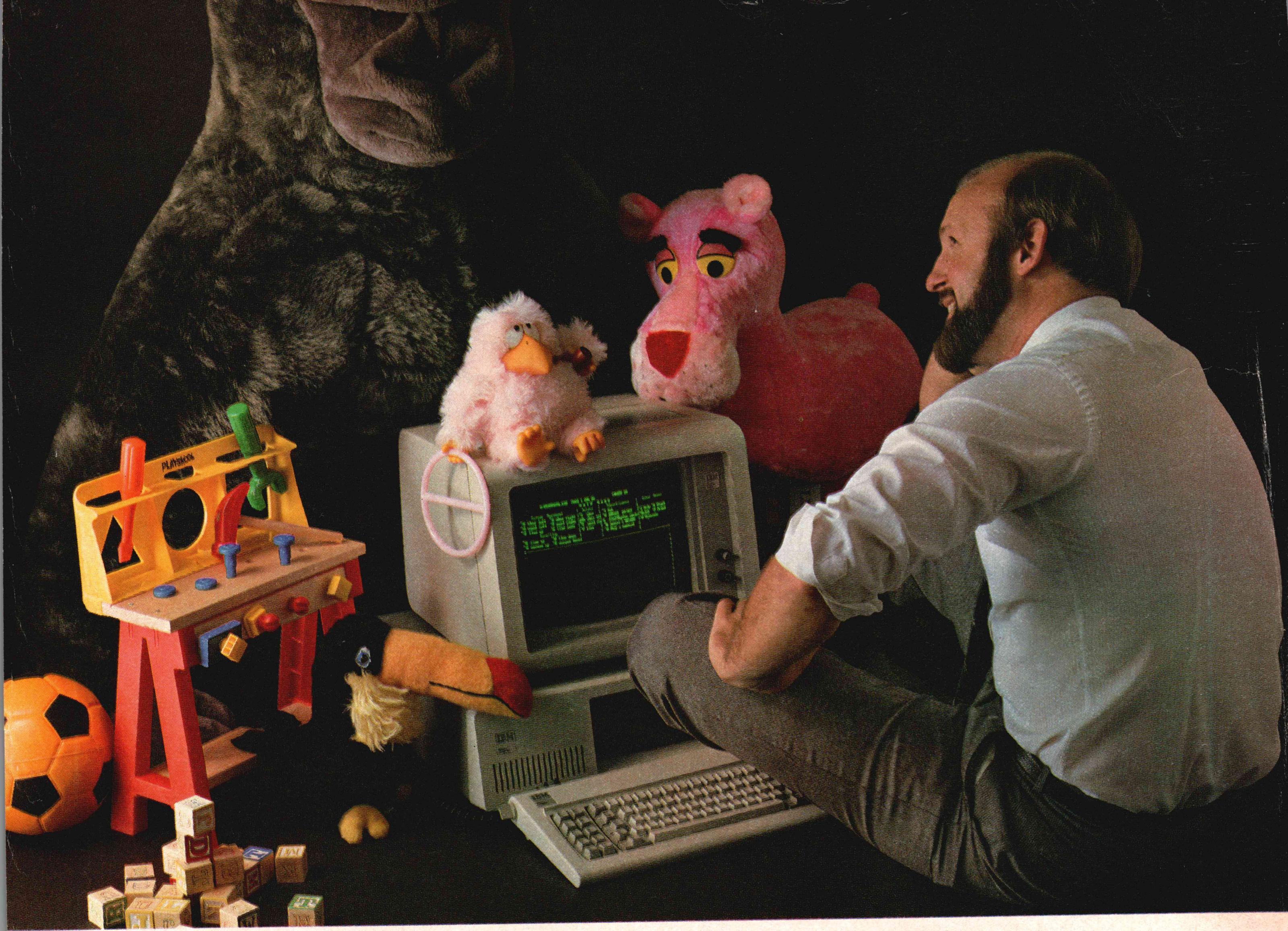
Binary Magic Numbers

8080 Fig-Forth Directory & File System

Forth Votrax Driver

. . . and More

DDJ T-Shirt pg. 45



This Programming professional deserves a lot more from his personal computer.

He's earned it. As a seasoned professional, he's learned to master some of the world's most advanced programming tools. Tools specially designed to meet the everyday demands of programming experts.

But as the owner of a personal computer, he's come to expect less. Less performance. Less sophistication. And less flexibility.

Why should programming a personal computer be any different?

Prior to the announcement of micro/SPF™ development software, experienced programmers felt programming a personal computer was a lot like playing with a toy. You couldn't take it seriously.

But today, there's micro/SPF,™ a solution to elementary program editing tools now offered with most micro-computers.

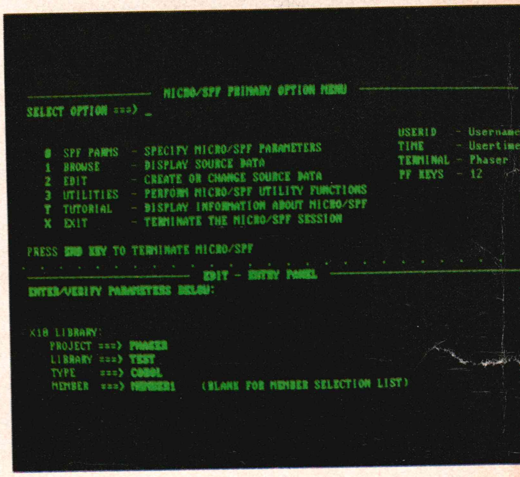
With micro/SPF™ you get the same procedures and commands experienced programmers are accustomed to using at work. By mimicking features found in standard SPF software, micro/SPF™ provides all the sophisticated utilities programming professionals expect.

Programming experts can take advantage of skills they've spent years perfecting.

Now, for the first time, mainframe software is available for personal computers. SPF screens are fully reproduced in logical sequence and each screen is formatted identical to those found in the SPF system.

In addition, micro/SPF™ comes equipped with the same primary and line commands, tutorial messages and program editor (with program function keys) experienced programmers are used to.

Programming professionals who've spent years perfecting the art of writing sophisticated code deserve to work with state-of-the-art tools, not toys. Find out how micro/SPF™ can help you do work-compatible programming on your personal computer today!



PHASER

PHASER SYSTEMS, INC 50 WEST BROKAW ROAD
SAN JOSE, CA 95110

Circle **no. 42** on reader service card.

Z-80® and 8086 FORTH

PC/FORTH™ for IBM® Personal Computer available now!

FORTH Application Development Systems include interpreter/compiler with virtual memory management, assembler, full screen editor, decompiler, demonstration programs, utilities, and 130 page manual. Standard random access disk files used for screen storage. Extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M	$ 50.00
8086 FORTH for CP/M-86	$100.00
PC/FORTH for IBM Personal Computer	$100.00

Extension Packages for FORTH systems

Software floating point	$100.00
Intel 8087 support (PC/FORTH, 8086 FORTH only)	$100.00
AMD 9511 support (Z-80, 8086 FORTH only)	$100.00
Color graphics (PC/FORTH only)	$100.00
Data base management	$200.00
Symbolic Interactive Debugger (PC/FORTH only)	$100.00
Cross Reference Utility	$ 25.00
Curry FORTH Programming Aids	$150.00
PC/GEN™ (custom character sets, IBM PC only)	$ 50.00

Nautilus Cross-Compiler allows you to expand or modify the FORTH nucleus, recompile on a host computer for a different target computer, generate headerless code, and generate ROMable code with initialized variables. Supports forward referencing to any word or label. Produces load map, list of unresolved symbols, and executable image in RAM or disk file. No license fee for applications created with the Cross-Compiler! Prerequisite: one of the application development systems above for your host computer.

Hosts: Z-80 (CP/M 2.2 or MP/M), 8086/88 (CP/M-86), IBM PC (PC/DOS or CP/M-86)
Targets: Z-80, 8080, 8086/88, IBM PC, 6502, LSI-11, 68000, 1802, Z-8

Cross-Compiler for one host and one target	$300.00
Each additional target	$100.00

AUGUSTA™ from Computer Linguistics, for CP/M 2.2	$ 90.00
LEARNING FORTH, by Laxen & Harris, for CP/M	$ 95.00
Z-80 Machine Tests Memory, disk, console, and printer tests with all source code in standard Zilog mnemonics	$ 50.00

All software distributed on eight inch single density soft sectored diskettes, except PC/FORTH on 5¼ inch soft sectored single sided double density diskettes. Micropolis and North Star disk formats available at $10.00 additional charge.

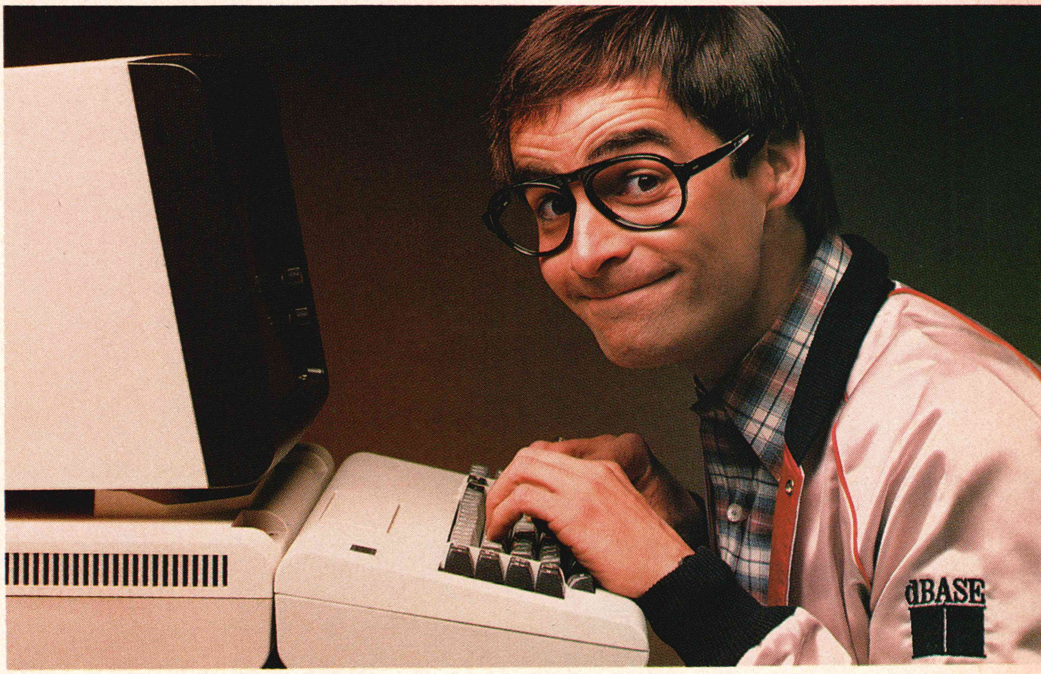
Prices include shipping by UPS or first class mail within USA and Canada. Overseas orders add US$10.00 per package for air mail. California residents add appropriate sales tax. Purchase orders accepted at our discretion. No credit card orders.

Laboratory Microsystems, Inc.

4147 Beethoven Street
Los Angeles, CA 90066
(213) 306-7412

Circle **no. 62** on reader service card.

All you dBASE II™ hotshots are about to get what you deserve.



You've written all those slick dBASE II programs.

Business and personal programs. Scientific and educational applications. Packages for just about every conceivable information handling need.

And everybody who sees them loves them because they're so powerful, friendly and easy to use.

But that's just not good enough.

Uh-uh.

Because now you can get the gold and the glory that you really deserve.

Here's how.

We've just released our dBASE II RunTime™ application development module.

And it can turn you into an instant software publisher.

The RunTime module condenses and encodes your source files, protecting your special insights and techniques, so you can sell your code without giving the show away.

RunTime also protects your margins and improves your price position in the marketplace. If your client has dBASE II, all he needs is your encoded application. If not, all you need to install your application is the much less expensive RunTime module.

We'll tell the world.

With your license for the dBASE II RunTime module, we provide labels that identify your program as a dBASE II application, and you get the benefit of all the dBASE II marketing efforts.

We'll also provide additional "how to" information to get you off and running as a software publisher sooner.

And we'll make your products part of our Marketing Referral Service. Besides putting you on our referral hotline, we'll publish your program descriptions and contact information in *dBASE II Applied*, a directory now in computer stores world-wide.

Go for it.

But we can't do any of this until we hear from you.

For details, write RunTime Applications Development, Ashton-Tate, 10150 West Jefferson Boulevard, Culver City, CA 90230.

Or better yet, just call (213) 204-5570. And get what you deserve today.

dBASE™ II

ASHTON·TATE

Dr. Dobb's Journal

For Users of Small Computer Systems

April 1983 Volume 8, Issue 4

Publisher/Art Director – Clifford West
Editor – Reynold Wiggins
Editorial Consultant – Marlin Ouverson
Contributing Editors –
Dave Caulkins, Dave Cortesi,
Ray Duncan, Gene Head,
Michael Wiesenberg
Marketing Director– Craig Harper
Marketing Manager – Beatrice Blatteis
Advertising Director – Carl Landau
Advertising Sales – Doug Millison
Circulation Manager – Gloria Romanoff
Circulation Assistants –
Terri Marty, Linda Marohn
Production Manager – Barbara Ruzgerian
Production Assistant – Jane A. McKean
Typesetter – Paula Fairchild
Cover Illustration – Al McCahon

Subscription Rates: $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

Writer's Guidelines: All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

Donating Subscribers **Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives *five* one-year subscriptions).

Contributing Subscribers: DeWitt S. Brown, Burks A. Smith, Robert M. Connors, Robert C. Luckey, Transdata Corporation, Mark Ketter, John W. Campbell, Friden Mailing Equipment, Howard E. Decker, Frank Lawyer, Rodney Black, John Brodie, Thomas Davis, Jan Steinman, Ronald E. Johnson, G. Hunter, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor, Jr., Ted A. Reuss III, Infoworld, Stan Veit, Western Material Control, S.P. Kennedy, Ed Moran. **Lifetime Subscriber:** Michael S. Zick.

Foreign Distributors **UK & Europe:** Homecomputer Vertriebs HMBH 282, Flugelstr. 47, 4000 Dusseldorf 1, West Germany; La Nacelle Bookstore, Procedure D'Abonnement 1-74, 2, Rue Campagne – Premiere, F-75014 Paris, France; Computercollectief, Amstel 312A, 1017 AP Amsterdam, Netherlands. **Asia & Australia:** ASCII Publishing, Inc., 4F Segawa Bldg. 5-2-2, Jingumae, Shibuya-Ku, Tokyo 150, Japan; Computer Services, P.O. Box 13, Clayfield QLD 4011, Australia; Computer Store, P.O. Box 31-261, 22B Milford Rd., Milford, Auckland 9, New Zealand. *(Write for Canadian distributors)*

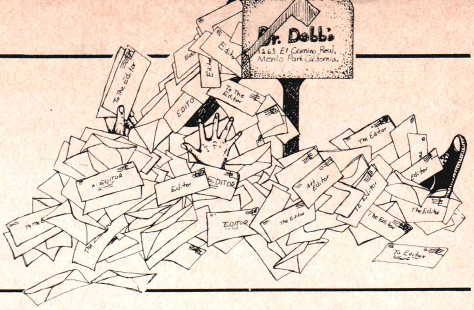
CONTENTS

ARTICLES

DEPARTMENTS

Dr. Dobb's Journal (USPS 307690) is published twelve times per year by People's Computer Company, P.O. Box E, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025 · 415/323-3111

LETTERS



Xanadu Re-Edited

Dear Dr. Dobb's:

Thank you very much for publishing our piece on the Xanadu Hypertext System in the January issue of *Dr. Dobb's*. While we were delighted to get word of our system in front of an audience, the editorial process somehow managed to introduce a couple of errors which deserve correction.

First of all, the article says that the Xanadu System is a currently available product. This is not entirely correct, and that was not how we wrote it. We *are* now marketing the system for large scale commercial applications, but we are doing so solely as a part of custom systems being produced on spec. While the system *does* work, it is as yet under development, and several more months of effort will be required before it will be a marketable product available "off the shelf." We are still working on various more advanced portions of the system, including the versioning facility and historical trace. We have been giving demonstrations regularly and are now looking for our first major customer. We are prepared to offer custom systems for applications such as engineering project management, software maintenance and source code management, and business document handling. We are currently quoting delivery times of six months to a year for such systems. Anyone interested should contact us for details, quotes or demonstrations.

Second, the article is credited to me. While I was the person who sent you the Xanadu material, the piece that you selected for publication, out of the two-hundred or so pages that we sent, was written by our Executive Vice President, Chip Morningstar. Since it was he who put in the hours writing it, he should get the credit for it.

We would like to talk with anyone interested in developing frontend systems. A preliminary protocol for frontend/backend interaction is available for the asking to anyone seriously contemplating frontend work. Our frontend development currently takes place on our SUN (trademark SUN Microsystems Inc.) workstation in C under UNIX (trademark Bell Laboratories). A copy of the current frontend is also available to anyone seriously interested. Most of our energies have been devoted to getting the backend working, and there are a lot of interesting human-machine interface problems yet to be attacked. Again, interested parties should contact XOC for details.

Thank you again.

Sincerely,
Roger Gregory, President
XOC
P.O. Box 7615
Ann Arbor, MI 48107

OK to Repeat

Dear Doc,

I'm writing this on the last day of 1982. It has been a couple of years since I last wrote, but a passing comment in J. H. Peters' letter in the January 1983 issue has moved me to the tyepwriter.

Mr. Peters' letter was specifically concerned with JRT Pascal but, as an aside, included the comment that "Pascal includes what is probably the worst programming practice known to modern science: running once through a loop before making any tests (REPEAT. . . UNTIL)." This comment pushed one of my hot buttons. That alone would not have sparked a letter except that this particular hot button has happened to be exercised repeatedly in the last few months.

REPEAT. . . UNTIL is not a programming practice at all. It is a tool provided by the language designer to be used, or not used, as may be deemed appropriate by programmers working in the language. This tool, like any other, may be left to rust in the toolbox or it may be used as needed when its use is deemed appropriate.

One good example of an appropriate use of this tool is provided by Kernighan and Ritchie in *The C Programming Language*. In their itoa(n,s) function, it is desired to convert an integer, n, to an ASCII

EDITORIAL

Welcome to our second ninety-six page issue! We are pleased to bring you more editorial pages than ever, and expect even more in the second half of 1983. As we grow, you will find more of the high-quality material that you have come to expect from *DDJ*. The increase in size will help ensure that the reader forum will continue to be provided in our pages. This is one of the things that separates us from the crowd – the difference between editors and experts.

* * *

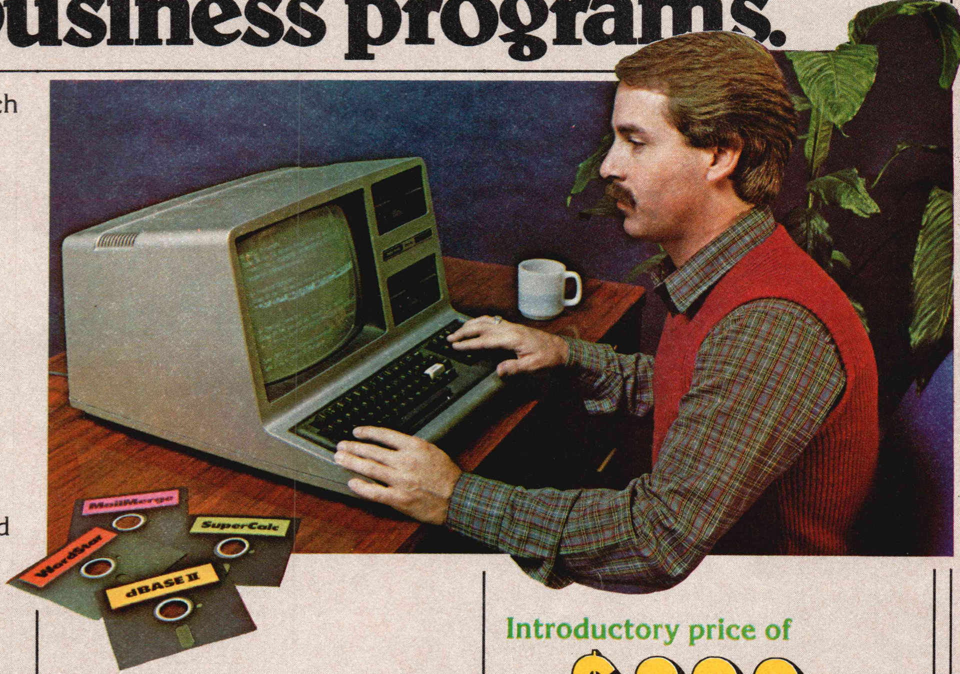
This issue's CP/M Exchange is the second of a two-part series on CP/M disk I/O. It marks Gene Head's last month as columnist for the Exchange, though he will continue to operate his RCP/M system. We would like to thank Gene for all his fine contributions over the past months and look forward to continued input from his Oregon Head Quarters. Gene is passing the pen to Bob Blum. No stranger to *DDJ* or to the Exchange, Bob provided the current series on disk I/O, and he will begin a series on CP/M Plus in the May issue. We are glad to welcome him aboard.

* * *

A couple of months ago we mentioned the possibility of condensing some listings in order to highlight exemplary or instructive sections of code. The motivation was to get *more* information to you, not less. We fear this may have given rise to the misinterpretation that we were making a significant shift toward eliminating listings. Not so! *Dr. Dobb's* has always been a forum for software and programming ideas. In keeping with our tradition, we will, of course, continue publishing complete listings. . . . Are those sighs of relief we hear?

Reynold Wiggins
Editor

Put 64K CP/M® 2.2 in your TRS-80 Model III and tap into 2,000 business programs.



Now you can run programs such as WordStar, dBASE II, SuperCalc, MailMerge and virtually thousands of other CP/M-based programs on your TRS-80 Model III.

CP/M 2.2 is the industry standard operating system that gives you access right now to over 2,000 off-the-shelf business programs.

Our plug-in Shuffleboard III comes with 16K of RAM, giving your Model III the power of full 64K CP/M 2.2 without interference of the ROM or video memory. In fact, the Shuffleboard will appear transparent in the TRS-80 mode and will not interfere with any DOS operation.

READ and WRITE Osborne, Xerox and IBM personal computer software plus many more popular formats.

Unfortunately, there is no standardized CP/M format for 5¼" diskettes. But we have developed a way to READ/WRITE and RUN standard programs under the following single-sided formats: Osborne 1 S/D, Xerox 820 S/D, IBM PC* D/D for CP/M 86 only, Superbrain D/D, Kapro II D/D, HP 125 D/D and TeleVideo D/D.

*Will Read and Write Only.

Easy plug-in installation.

It's so simple. The Shuffleboard III plugs into two existing sockets inside your Model III. There are no permanent modifications, no cut traces and no soldering. You'll be up and running in minutes.

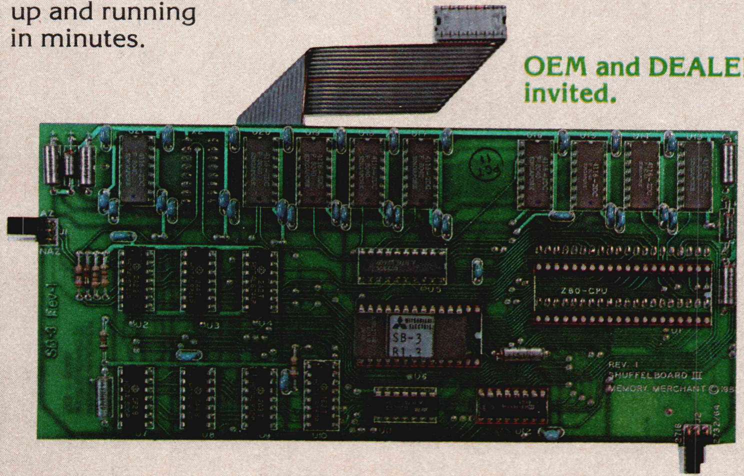
New Products.

80 x 24 VIDEO BOARD: Features dual intensity screen, programmable cursor control for block, underline & blink rate, on-board bell with audible keyclick, battery-operated real time calendar/clock, full ASCII character set plus 256 special character graphics, dual RS-232 outputs and composite video output.

FLOPPY DISK CONTROLLER: Now you can access 5¼" and 8" floppy disk drives in any combination up to 4 drives of S/D density, S/D sided. Tap into a wealth of CP/M software which comes on 8" IBM 3740 format or Pickles & Trout CP/M for the Model II.

SOFTWARE: Additional CP/M software programs are available. Call or write for details.

OEM and DEALER inquiries invited.



Introductory price of

$299.

The Shuffleboard III comes fully burned-in and tested complete with 64K CP/M 2.2 and MBASIC 80 interpreter, plus software manuals and a first class user's manual — with a 1-year limited warranty and 15-day no-risk free trial — for only $299.

See the Shuffleboard III at your dealer's now.

Once you see what the Shuffleboard can do for your Model III you'll want one at once. If your dealer does not yet stock the Shuffleboard have him give us a call. Or send check, money order, VISA or MASTERCARD number (sorry, no COD's) plus $5 shipping per board ($17 outside the USA & Canada)* directly to the address below. Cal. residents please add sales tax. Credit card purchases can be phoned in directly and we'll ship from stock.

(415) 483-1008

*Air mail shipments to Canada & all other countries.

Memory Merchant™

14666 Doolittle Drive San Leandro, CA 94577
(415) 483-1008

Circle **no. 54** on reader service card.

string at s. Leading zeroes are to be suppressed *unless* n has a value of zero, in which case a single ASCII zero should be generated. This exceptional case is neatly accommodated by passing once through the loop before testing for a zero value in n.

Best regards, and best wishes in the new year.

William T. Mitchell
P.O. Box 94978
Schaumburg, IL 60194

Hardware Help

Gentlemen,

I would like to see a hardware article on making a dynamic RAM board using an Intel 8203 dynamic RAM controller (for 64K DRAMs) or 8207 (for 256K DRAMs). Intel has application notes on just such topics, but they provide only block diagrams (all of the ICs are listed) and I'd like to see a complete schematic of a working board.

I've been looking for information on connecting a Model 33 teletype to my TRS-80 Color Computer and my Sinclair ZX81. I've been able to find a general RS232 to 60mA current loop interface diagram, but would like more specific information (my Model 33 TTY is set for 20mA current loop). I'm sure many people have used Model 33 TTYs as printers, but I've been unable to locate any (sorry this note isn't printed!). I'd greatly appreciate any help you could give me on this matter (both hardware and software).

Yours truly,

Timothy J. McIlwee
Rural Route 2, Box 462A
Dundee, IL 60118

Forth Loader Fix

Dear *Dr. Dobb's*:

I encountered some problems using the Forth Relocating Loader which was published in your September 1982 issue (No. 71). Upon examination, I found that the program was doing signed comparisons of addresses instead of unsigned comparisons (i.e., using < and > instead of U< and U>). I corrected this in screens 10 and 11, and in 14 through 19, and everything worked fine after that.

Thanks, and keep up the good work on your magazine.

David J. Sieber
Star Computer Systems
20600 Gramercy Place
Torrance, CA 90501

DIR.ASM Problems?

Dear *DDJ*:

DIR.ASM, published in the CP/M Exchange column in February 1983, probably should not have been published at all; but since it was, it should have had the warning: "Use this program at your own risk. It is untested."

I wrote the program in response to a request to give the readers code. After hurriedly submitting the untested program, I found it to be machine dependent in some cases and told the editor about the problems. I was assured a disclaimer would be included, and was told that the code was a good starting point for a better directory utility and maybe a challenge to hackers to get operating on specific systems.

Since, then, I found SD-41.ASM in the public domain. This is a far better directory utility than DIR.ASM with more features and faster execution not to mention fewer (zero) known bugs.

To those trying to get DIR.ASM working, I suggest you scrap it and get a copy of SD-41.ASM (and SD-41.DOC) from any RCP/M system. I apologize for any inconvenience and in the future will not make the mistake of releasing code without complete field testing.

Sincerely,

Gene Head
Head Quarters
2860 NW Skyline Drive
Corvallis, OR 97330

Hi-Res Vector Display Revisited

Dear *DDJ*:

Following my letter in the September 1982 issue (*DDJ* No. 71), I have received many letters concerning the color graphics display I described therein. I cannot spare the time to give individual replies but do intend to write up the system for publication when it is finally perfected. There is still a lot of work to do before it can be confidently offered to others to duplicate, mainly in cutting the enormous cost – e.g., the special CRT used costs over $1000 on a one-off basis.

Have other 68000 users any ideas on standardization of memory maps, disk formats, etc. to allow software interchange? There seem to be no standards at all at the moment and I fear we will have a similar situation to that which now applies to the 6502, with many machines having no interchangeability of software whatever. My own choice, for what it's worth, is to place ROM between 000000 and 00FFFF, I/O ports from 010000 to 01FFFF and RAM in the rest of the address space. I use the IEEE-696 S-100 bus, which is very suitable for this system and gives good hardware interchangeability – I have processor boards with other CPUs which I can use when I need to.

Finally, does anyone have any information concerning a Wangco 8201 floppy disk controller board and an MFE model 250B digital cassette deck? Wangco no longer make the former and cannot supply any information and I've no idea who MFE are at all. Any info would be very welcome.

Sincerely,

Greg Trice
1131 Sandhurst Circle, #111
Scarborough, Ontario
Canada M1V1V5

Ruzinsky Re-Corrected

Dear Readers,

Steven Ruzinsky wrote in to tell us that a production error had caused an incorrect correction to his equations on page 10, column 3 of the March issue. Lest any be confused by the resulting "non-equation," the text *should* have read:

```
Let U = Angle
S1 = .9999999999999999
V1 = S1*[FABS(FABS[FPREM
        (U,2*PI)] - PI) - PI/2]
COS(U) = Y(V)/FSQRT(X(V)*X(V)
        +Y(V)*Y(V))
```

Our apologies to Mr. Ruzinsky and anyone who was inconvenienced.

The Editor

DDJ

CP/M EXCHANGE

by Gene Head

Bob Blum will take over this column beginning next month. Bob has brought considerable technical expertise with this two-part series on the CP/M disk and plans to begin a series on CP/M+ next month.

Take the time to read and re-read this month's column. It's all meat and no fat but you may need a pencil and paper to do some quick calculations. It was a real eye-opener for me. Bob knows his stuff!

Last month we reviewed how CP/M allocates and maintains disk space. Our discussion this month will continue by exploring first the file control block (FCB) and then the various BIOS tables used to describe the disk system and their interaction with CP/M.

When accessing a disk file through BDOS, a properly initialized FCB must be used. The open routines use the file name and type fields as the search argument for the desired file. Upon a successful match, the allocation information from the stored extent is copied into the FCB for future file operations. During input/output (I/O), various fields of the FCB are used to maintain an updated copy of the file's status.

When writing to a file, if the FCB's addressing range is exceeded or the file is closed, the current FCB is written to the disk directory after the last four bytes are discarded. This truncation maintains compatibility with earlier releases of CP/M and also eliminates unneeded information. It should be noted that the names, FCB and extent, can be used interchangeably, although FCB typically refers to the memory resident file control block, while extents are disk resident.

Referring to Figure 1, the first byte of the FCB, DR, has a dual purpose depending on whether the FCB is memory or disk resident. When the FCB is in memory, the DR entry (when non-zero) designates which disk drive is to be auto-selected. After the FCB is written to disk, the DR entry reflects the file user number. The next 11 bytes contain eight bytes for the file name and three bytes for the file type. The high-order bits of the first two bytes of the file type are used to indicate if the file is read-only or a system file. Only two of the next three bytes are important to this discussion. The first byte, EX, contains the current extent number in the range of 0 to 31. If the file expands beyond 32 extents, the EX byte is zeroed and the third byte, EC, is incremented. These two bytes can be visualized as containing a nine-bit number with the EX byte containing the low-order five bits and the high-order four bits present in the EC byte. To verify the maximum file size of 8,388,608, multiply each extent of 16,384 bytes by 512 extents.

Another way to view the association between these two fields is that the EX byte contains the extent number within a group contained in the EC byte. Continuing along to the remaining fields, as each 128-byte sector is written, the record count byte, RC, is incremented by one until 128 is reached. When this occurs, the FCB is written into the directory and a new FCB is automatically prepared for additional file operations. We can now verify that each extent can address 16,384 bytes by multiplying 128 sectors by 128 bytes per sector. The allocation map area of the FCB, AM, provides room for 16 bytes of data allocation group numbers. Each allocation group number can be either one or two bytes in length depending on how many allocation groups are contained on the disk. Further attention will be given to this later in the discussion on disk parameter blocks. The remaining four bytes of the FCB are used for sequential and random access into a file and are left for independent study.

As delivered by Digital Research, CP/M is configured for the MDS-800, which is not in common use. For that reason we will assume that CP/M is unconfigured as distributed. Before studying how to configure CP/M for a particular disk system it may be helpful to examine an overview of all the integral parts of the BIOS. Figure 4 (page 12) shows us that the CBIOS begins with a series of sixteen jumps followed by one disk parameter header (DPH) for each disk drive attached to the system. The DPH contains address pointers to various other tables that completely describe the disk formats that can be used on this particular system.

From Figure 2 (page 10), the first field in the DPH, XLT, is a two-byte pointer to

FILE CONTROL BLOCK

OFFSET	SYMBOL	BYTES	DESCRIPTION^S
0	DR	1	Drive number: 0 = Default drive 1 = Auto select drive A ----- ----- 16 = Auto select drive P
1	FN	8	File name in upper case ASCII. Padded on right with blanks if necessary.
9	FT	3	File type in upper case ASCII. High order bits contain additional information. BYTE 9 = 1 Read only file BYTE 10 = 1 System file. File will not be listed during a directory list.
12	EX	1	Number of current extent in use.
13	WA	1	Workarea.
14	EC	1	When extent number, EX, exceeds 32 this byte acts as an overflow area.
15	RC	1	Record count. Number of 128 byte records contained in this extent.
16	AM	16	Allocation map. Allocation group number storage area.
32	CR	1	Current record. Number of record currently be accessed.
33	RR	3	Random record. Record number to be used for accessing the file randomly.

Figure 1.

the physical sector translation table. Through this table it's possible to translate a logical sector to its physical equivalent. Many of today's systems allow different disk formats to be used interchangeably on the same drive. For this reason, when logging in a disk, the BIOS will probably access a known part of the disk to determine the format and place the address of the corresponding disk parameter block into DPB and the appropriate translation table address into XLT.

The next two bytes are used for a workarea by BDOS followed by a two-byte pointer, DIRBUF, to an area of memory 128 bytes long which is used for directory operations. All DPHs can point to the same DIRBUF because only one directory operation will be active at a time. For example, when a file is opened, the directory is read into the buffer pointed to by DIRBUF one sector at a time. If the desired file is found, an index number is returned to the requester so that additional information can be extracted. CSV points to a workarea used for calculation of the directory checksum. This checksum is used to detect when a disk has been changed without a warm start. Each disk drive must have its own CSV workarea. The DPH ends with a two-byte pointer, ALV, to a savearea which will contain the allocation map after a disk is logged in. A separate ALV savearea must be reserved for each disk drive on the system.

The disk parameter block (see Figure 3) contains all the information necessary to describe the disk format in use. The first entry, SPT, contains the total number of logical sectors per track. This number may not equal the number of physical sectors actually written on each track. For example, TARBELL double-density format uses sixteen 512-byte sectors per track. Since there are four 128-byte sectors per physical sector, SPT would contain 64.

Before discussing the next three fields, a few decisions must be made. First, decide how large the data allocation block should be. On systems where the total capacity of any one disk is less than 225K, block sizes from 1K to 16K can be used. On larger capacity formats you are limited to selecting block sizes 2K and larger. Tradeoffs between block sizes are subtle. If you consistently create small files, a smaller block size will probably be best suited to your needs. Larger block sizes have the advantage of keeping more sequential data available without excessive head movement.

Next determine how many data blocks can be allocated. This is a function of the total capacity of the disk format being used minus the number of reserved system tracks divided by the data block size. This value is placed in DSM.

With these decisions made, the tables in Figure 5 (page 12) are used to provide

DISK PARAMETER HEADER

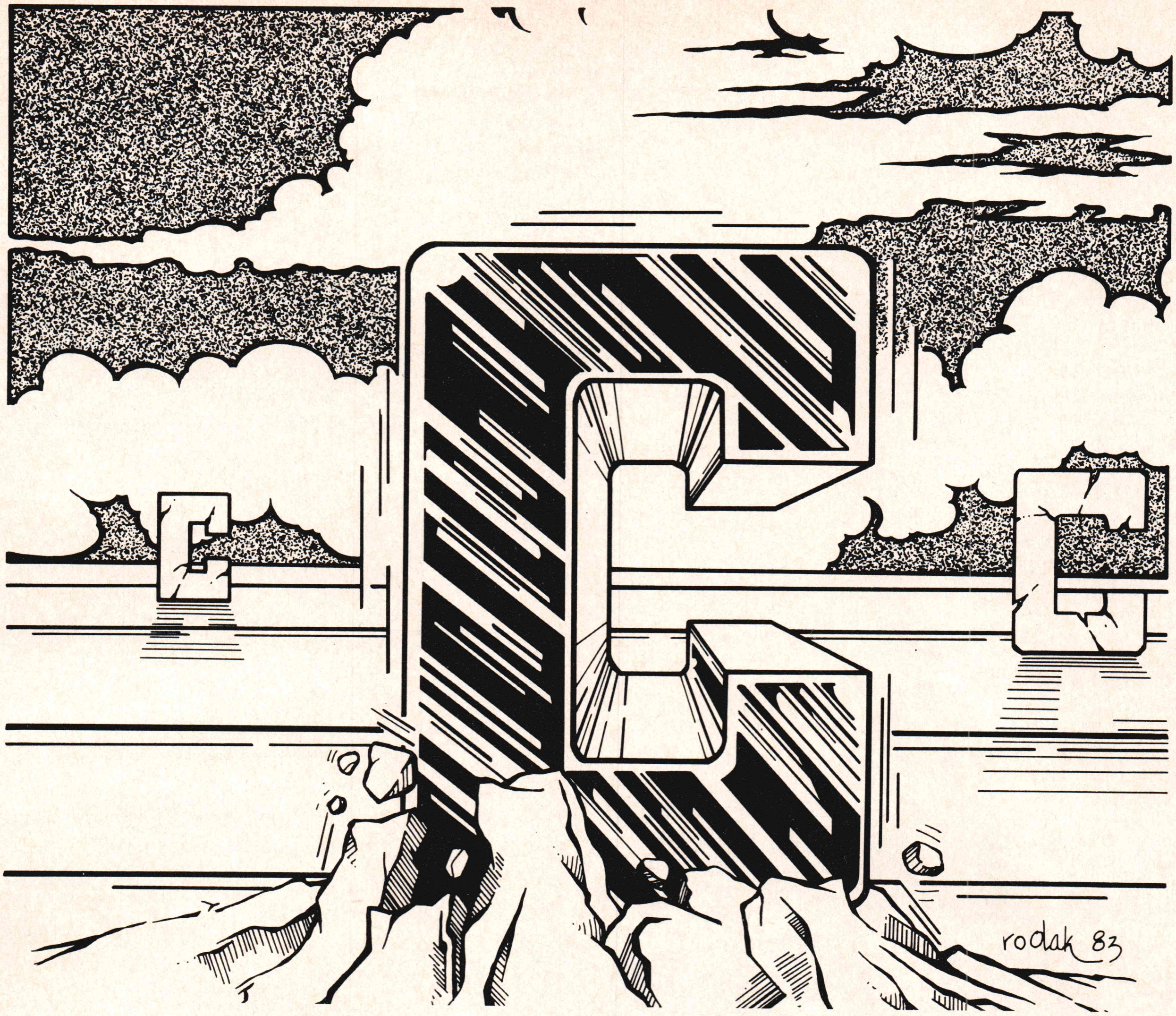
OFFSET	SYMBOL	BYTES	DESCRIPTION^S
0	XLT	2	Address of the logical to physical sector translation table.
2		6	BDOS workarea.
8	DIRBUF	2	Address of a one sector workarea used for directory operations.
10	DPB	2	Address of the disk parameter block for this drive.
12	CSV	2	Address of a workarea used to calculate the directory checksum to detect changed disks.
14	ALV	2	Address of a savearea used for this drives allocation map.

Figure 2.

DISK PARAMETER BLOCK

OFFSET	SYMBOL	BYTES	DESCRIPTION^S
0	SPT	2	Total number of 128 byte sectors per track.
2	BSH	1	Data allocation block shift factor as determined by the data block allocation size.
3	BLM	1	Data allocation block mask.
4	EXM	1	Extent mask determined by the number of data blocks allocated and the data allocation block size.
5	DSM	2	Total number of data blocks allocated to this drive.
7	DRM	2	Total number of directory entries allocated for this drive minus 1.
9	AL0 AL1	1 1	Table of 16 bits set on from left to right indicating allocation groups reserved for directory usage.
11	CKS	2	Size of the directory check vector.
13	OFF	2	Track offset to logical track zero.

Figure 3.


rodak 83

values for BSH, BLM, and EXM. The extent mask value further redefines how much storage can actually be addressed by each FCB. For example, disks with less than 255 data allocation groups only require one byte for each group pointer. If the allocation groups are 2K in size then only eight bytes are used for each 16K addressed leaving eight bytes of allocation map unused in the FCB. By coding EXM with 1, two logical extents can be stored in each physical FCB extending its addressing range to 32K.

The situation becomes more confused when there are more than 255 allocation groups. In this case each pointer requires two bytes. In any case the tables in Figure 5 will provide the proper entries to be used. The total number of directory entries, DRM, is also a function of the data block size. Each directory entry is 32 bytes long. If the data block size is 1K then 32 directory entries will be contained in each block.

Typically two data blocks are reserved for directory use. DRM would then be coded with 63. The decision on how many data blocks to reserve for the directory is reflected in entries AL0 and AL1. Given that two data blocks are reserved for directory use, the two high-order bits of the 16-bit quantity formed by combining AL0 and AL1 would be set on. This value is used when a new disk is logged in to mask the allocation vector table to prevent any damage to the directory data blocks. The next entry, CKS, should contain the total number of directory entries in use. Each time a disk is selected after it has been logged in, the directory is checked against the stored checksum to determine if the disk has been changed without a warm start. If a hard disk is in use it is not necessary to provide for directory checking.

By using the information contained in the disk parameter block, you can move about the disk system at will. To calculate the starting track and sector numbers from a known allocation group number requires only a few steps. First multiply the group number by BLM plus one. This can easily be done by shifting the group number left the number of times contained in BSH. This results in supplying the actual sector number as if all the sectors were sequentially numbered. Now divide the result by SPT and add OFF to the quotient. The remainder is the starting sector number while the quotient is the starting track number.

To calculate a group number with known track and sector numbers does not require any more effort. Subtract OFF from track then multiply the track number by SPT and add the sector number to the quotient. Now divide the quotient by BLM plus one.

In a very short time we have covered the same material that has been presented in a number of books. I suggest that you use the information presented here while reviewing a BIOS listing. Next month the features of CP/M Plus will be compared to those provided by CP/M 2.x.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 231**


```
                    BIOS CONSTRUCTION

JMP        COLDSTART
JMP        WARMSTART
      ------
      ------
JMP        LISTSTAT
DB         (Disk Parameter Header for Drive 1)
DB         (Disk Parameter Header for Drive N)
           X     X     X     X     X
           X     X     X     X     X
XXXXXXXXXXX      X     X     X     X
X             XXXX     X     X     X
XLT             X     DPB   CSV   ALV
          DIRBUF
```

Figure 4.

DETERMINING BSH - BLM UNITS

BLS	BSH	BLM
1024	3	7
2048	4	15
4096	5	31
8192	6	63
16384	7	127

BLS	EXM
1024	0
2048	1
4096	3
8192	7
16384	15

If DSM > 255

BLS	EXM
1024	N/A
2048	0
4096	1
8192	3
16384	7

Figure 5.

You've spent thousands on your system.

Will you invest two dollars to make the most of it?

OF COURSE YOU WILL.*

Computers: A Comprehensive Book Guide is a 64-page annotated mail order catalog of the best microcomputer books published to date. It critically reviews 819 books chosen to help you make the most of your microcomputer. **Computers: A Comprehensive Book Guide** is organized into 26 topics including:

- Business Applications
- Word Processing
- Assembly language & microprocessors
- Machine specific hardware & software
- Programming languages

TRY our fast, efficient, personal mail order service. The **Yes! Bookshop**, established 1970, stocks all the books reviewed. We also welcome special orders.

AND as a **Yes! Bookshop** customer, you will receive free updates, reviewing the latest microcomputer books as they are published.

WHO SAID MICROCOMPUTERS WOULD MAKE BOOKS OBSOLETE?

*Here's my $2 (refunded with first order).
Please send **Computers: A Comprehensive Book Guide** to:

Name ____________________

Address ____________________

____________________ Zip

Yes! Bookshop
1035-A1 31st Street N.W.
Washington, D.C. 20007

Mail Orders: (202) 338-2727
M-F: 9-5
VISA, MC, AMEX

Circle **no. 17** on reader service card.

RECLAIM

A File Reclamation Utility for Destroyed Directories

Editor's note: The author informed us that he wrote this utility in self-defense, after an errant program had destroyed his directory of a megabyte of important files. It took him about fifty man-hours, including the writing of the program, to completely recover all those files. The source code for this utility is written for use with Ron Cain's version 1 of the Small-C compiler. It has not yet been tried with J. E. Hendrix's upgrade, Small-C version 2 (see DDJ *Nos. 74 and 75), but it should be upwardly compatible.*

This utility is available from the author on 8-inch, SSSD, CP/M-format diskettes for $30, which include the source code, additional documentation on use and configuration, and an executable version (for CP/M 2.2). For other formats, send inquiries to the author.

Many people who make frequent use of microcomputers have had the disheartening experience of loss of data in a disk system. This can be due to operator error, program error, or hardware malfunction. Experienced people usually will not offer any sympathy to those who lose important data because it should have been backed up on some removable media. Despite this commonsense rule, people still fall prey to various time and productivity pressures and occasionally lose data without having made a backup. In some cases the data lost can be extensive and valuable and warrant an effort in recovery. This is why disk patching programs exist; many can be found in the commercial marketplace and the public domain. Mostly they are useful if files have been inadvertently erased, or a sector develops an error and becomes unreadable due to a variety of possible reasons. Erased files can be reclaimed if the disk is not subsequently written to because erasing merely puts a flag in the directory entry for that file which says the file is erased – the data is not literally erased. Therefore that flag can be changed and the file "un-erased."

by Walter V. Murphy

Walter V. Murphy, Compucations, 212 Northwood Avenue, So. San Francisco, CA 94080.

The problem is on a more serious level, however, if the directory is more extensively damaged. In a CP/M-based system, storage space on a disk is allocated in blocks. A block is always a multiple of a sector, typically 8, 16, or 32 corresponding to 1K, 2K, or 4K byte units of allocation where a sector contains 128 bytes. These blocks are given sequential reference numbers and these numbers are placed in a list in the directory and associated with a given file name. The blocks assigned to a file need not be contiguous and it is permitted for them to be scattered in any order on the disk. These block numbers are the single most important piece of information the operating system deals with in maintaining a file. Clearly if the list of block numbers is destroyed for an individual file, the problem of reclamation is worsened, but if the directory for an entire disk is totally destroyed, the magnitude of the problem can be overwhelming, particularly if the disk was large and almost full.

The program presented here deals with this kind of problem. The reclamation process cannot be fully automated; a qualified person familiar with the types of data on the disk must still inspect the individual blocks and determine which to save and the order in which to group them. This inspection process requires one of the sector examination programs. One which I have used is DISKDOC, written in C and published in *DDJ* No. 66. It would make a nice companion to RECLAIM. The next step is to retrieve these multiple-sector blocks from the disk and write them into a file on another disk. RECLAIM is almost indispensible at this step in all but the most trivial cases.

How RECLAIM Works

RECLAIM is written in Small-C v.1 and intended for use on CP/M-compatible operating systems. It is run while logged into the disk which is to be reconstructed and it writes out files containing a single block each onto a different disk. RECLAIM, as it is presented here, can only be run on systems compatible with CP/M 2.0 or better. This is because RECLAIM gets specific file system information from the disk parameter block. However, it could be modified to get the file system information directly from a CP/M 1.4 system. RECLAIM makes direct BIOS calls in order to fill a block-sized buffer with all the sectors of the requested block and then uses the file system write command to create the file. When RECLAIM is run, it expects input from STDIN, the standard input device, which can be a file using Small-C's directed console input. This input is one text line for each block to be extracted from the disk. Each line contains the track number, the starting sector number of the block, and the filename including drive specifier in which to write the block. Whenever a large quantity of blocks is to be reclaimed, the list of blocks should be put into a text file and presented to RECLAIM through console redirection. The RECLAIM program and the block list file should never be written to the disk which is to be reclaimed; instead they should be accessed from another disk by using drive specifiers. The following example should be very clarifying.

The block list file "INPUTFIL" is a text file on A: containing:

```
2,17,a:file.01
2,33,a:file.02
2,49,a:file.03
5,33,a:file.04
23,47,a:file.05
54,1,a:file.06
```

The damaged disk is in drive B:. RECLAIM is on drive A:. Drive B: should be logged in as shown in Figure 1 (page 16). If at any point in the process an error occurs, the program trys to give appropriate error messages and then stops.

The overall recovery process can be summarized as follows:

(1) Use a disk sector examination program such as DISKDOC to inspect individual sectors for data characteristics. For example, it should be very easy to differentiate .COM files and text files by inspection as most disk sector examination programs also show the ASCII test equivalent of the bytes whenever possible.

(2) Decide which blocks will be kept and which are unimportant. Make a list of blocks by track and starting sector number. It is not necessary to include every sector you wish to save, only the starting sector of every block. The entire block will be automatically retrieved by RECLAIM. The block structure starts on your disk at the next sector immediately after the directory. In a typical double-density system with 64 directory entries, the first block is on track 2, sector 17 and blocks are 16 sectors long. At this time make any notes you can identifying the contents of the block so it will be easier

68000

CROSS ASSEMBLER

FOR CP/M−80 $ 260

MOTOROLA SYNTAX **MACROS**

LINKAGE EDITOR **CONDITIONALS**

STRUCTURED **WRITTEN IN C**

Quelo
843 NW 54th
Seattle, Wa. 98107

(206)784-8018
mornings
Dick Curtiss

CP/M is a trademark of Digital Research

Circle **no. 38** on reader service card.

MicroScript™

Are you wasting valuable time trying to format complex documents with a word processor or obsolete text formatter?

MicroScript™ is a state of the art text formatter specifically designed for the production of technical manuals, specifications, and other complex documents. This powerful tool pays for itself the first time you use it. Featuring:

- generalized markup
- left alignment
- center alignment
- right alignment
- justification
- left indention
- right indention
- bold text
- underscored text
- proportional spacing
- fully definable page
- multiple columns
- headers and footers
- floating text blocks
- footnotes
- variable line spacing
- widow supression
- section numbering
- imbedded documents
- automatic lists
- macro processing
- symbol processing
- table of contents
- direct printer control
- initialization profile
- page numbering

$99 postpaid within U.S., outside U.S. add $10. CA residents add 6%. Specify CP/M-80*, CP/M-86*, MS-DOS*, or PC-DOS*; printer type; disk format.

Software Technique™
6531 Crown Blvd., Suite 3A
San Jose, CA 95120
(408) 997-5026

CP/M-80, CP/M-86 trademarks of Digital Research, MS-DOS trademark of Microsoft, PC-DOS trademark of IBM Corporation.

Circle **no. 31** on reader service card.

edward k. ream

PRESENTS

RED

A TEXT EDITOR IN C

- **available for small-C and BDS C (specify when ordering)**
- **complete SOURCE CODE provided**
- **handles huge files**
- **block move and copy commands**
- **works with any video terminal with cursor addressing**
- **supplied on single density, IBM format, 8 inch disks for CP/M systems with at least 48K memory**
- **portable to other machines and operating systems**

Price: $50.

to order, or for more information, contact:

Edward K. Ream
1850 Summit Ave.
Madison, WI 53705
(608)231-2952

Circle **no. 35** on reader service card.

later to combine the blocks in a way that makes sense.

(3) Using a text editor, create a file which contains the list of blocks you wish to reclaim.

(4) Run the RECLAIM program while directing the input text file into it.

(5) Examine any of the files created which are text and concatenate them appropriately while giving them meaningful names. If some files are purely data then you will need information on the structure of that data in order to piece it back together. In certain databases it is not unusual to find a field which represents the date, a stock number, or line item number. Usually there is either a fixed record length for the database or a consistent identifying pattern at the end of each record such as a carriage-return line-feed pair. Knowing this and using a debugger such as DDT, you can decide which blocks are associated with each other. Later, blocks can be concatenated – using your text editor if they are text, or using PIP if they are any other kind of data. Text data can be verified by printing it out and looking at its appearance. Source code to programs can be verified by assembling or compiling them and database files can be verified by using them in a test application.

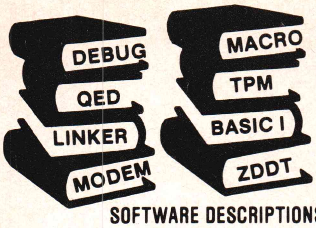
I think it is good advice to not attempt to erase or in any way modify the disk being reconstructed until you have not only retrieved all the blocks of interest but are also confident that the order in which you have restored them is correct. Good luck if you attempt a large recovery procedure of this sort. Although it is tedious and time consuming, you have nothing to lose. The best advice is, as always, keep backups of all important data and avoid ever having to reconstruct it.

```
B>
B> a:reclaim <a:inputfil
     COMPUCATIONS Disk block RECLAIMation utility, 12Aug82.
     This disk has 54 sectors per track and 16 sectors per block.
     Track: 2, Starting Sector: 17, Filename: a:file.01
     Track: 2, Starting Sector: 33, Filename: a:file.02
     Track: 2, Starting Sector: 49, Filename: a:file.03
     Track: 5, Starting Sector: 33, Filename: a:file.04
     Track: 23, Starting Sector: 47, Filename: a:file.05
     Track: 54, Starting Sector: 1, Filename: a:file.06
     Successful completion. Goodbye from RECLAIM.
B>
```

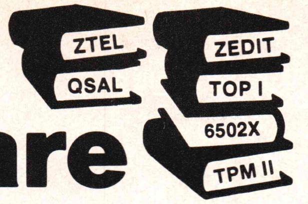
Figure 1.
RECLAIM Log On for Drive B:

(Listing begins on page 18)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 233**



Z80 Software



SOFTWARE DESCRIPTIONS

TPM (TPM I) - $80 A Z80 only operating system which is capable of running CP/M programs. Includes many features not found in CP/M such as independent disk directory partitioning for up to 255 user partitions, space, time and version commands, date and time, create FCB, chain program, direct disk I/O, abbreviated commands and more! Available for North Star (either single or double density), TRS—80 Model I (offset 4200H) or II, Versafloppy I, or Tarbell I.

TPM-II - $125 An expanded version of TPM which is fully CP/M 2.2 compatible but still retains the extra features our customers have come to depend on. This version is super FAST. Extended density capability allows over 600K per side on an 8" disk. Available preconfigured for Versafloppy II (8" or 5"), Epson QX-10, Osborne II or TRS-80 Model II.

CONFIGURATOR I

This package provides all the necessary programs for customizing TPM for a floppy controller which we do not support. We suggest ordering this on single density (8SD).
Includes: TPM-II ($125), Sample PIOS (BIOS) SOURCE ($FREE), MACRO II ($100), LINKER ($80), DEBUG I ($80), QED ($150), ZEDIT ($50), TOP I ($80), BASIC I ($50) and BASIC II ($100)
$815 Value **NOW $250**

CONFIGURATOR II

Includes: TPM-II ($125), Sample PIOS (BIOS) SOURCE ($FREE), MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), QSAL ($200), QED ($150), ZTEL ($80), TOP II ($100), BUSINESS BASIC ($200) and MODEM SOURCE ($40) and DISASSEMBLER ($80)
$1485 Value **NOW $400**

MODEL I PROGRAMMER

This package is only for the TRS-80 Model I. Note: These are the ONLY CDL programs available for the Model I. It includes: TPM I ($80), BUSINESS BASIC ($200), MACRO I ($80), DEBUG I ($80), ZDDT ($40), ZTEL ($80), TOP I ($80) and MODEM ($40)
$680 Value **NOW $175**

MODEL II PROGRAMMER

This package is only for the TRS-80 Model II. It includes: TPM-II ($125), BUSINESS BASIC ($200), MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), QED ($150), ZTEL ($80), TOP II ($100), ZDDT ($40), ZAPPLE SOURCE ($80), MODEM ($40), MODEM SOURCE ($40) and DISASSEMBLER ($80)
$1445 Value **NOW $375**

BASIC I - $50, a 12K+ basic interpreter with 7 digit precision.

BASIC II - $100, A 12 digit precision version of Basic I.

BUSINESS BASIC - $200, A full disk extended basic with random or sequential disk file handling and 12 digit precision (even for TRIG functions). Also includes PRIVACY command to protect source code, fixed and variable record lengths, simultaneous access to multiple disk files, global editing, and more!

ACCOUNTING PACKAGE - $300, Written in Business Basic. Includes General Ledger, Accounts Receivable/Payable, and Payroll. Set up for Hazeltine 1500 terminal. Minor modifications needed for other terminals. Provided in unprotected source form.

MACRO I - $80, A Z80/8080 assembler which uses CDL/TDL mnemonics. Handles MACROs and generates relocateable code. Includes 14 conditionals, 16 listing controls, 54 pseudo-ops, 11 arithmetic/logical ops, local and global symbols, linkable module generation, and more!

MACRO II - $100, An improved version of Macro I with expanded linking capabilities and more listing options. Also internal code has been greatly improved for faster more reliable operation.

MACRO III - $150, An enchanced version of Macro II. Internal buffers have been increased to achieve a significant improvement in speed of assembly. Additional features include line numbers, cross reference, compressed PRN files, form feeds, page parity, additional pseudo-ops, internal setting of time and date, and expanded assembly-time data entry.

DEVELOPER I

Includes: MACRO I ($80), DEBUG I ($80), ZEDIT ($50), TOP I ($80), BASIC I ($50) and BASIC II ($100)
$440 Value **NOW $150**

DEVELOPER II

Includes: MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), BUSINESS BASIC ($200), QED ($150), TOP II ($100), ZDDT ($40), ZAPPLE SOURCE ($80), MODEM SOURCE ($40), ZTEL ($80), and DISASSEMBLER ($80).
$1280 Value **NOW $350**

DEVELOPER III

Includes: QSAL ($200), QED ($150), BUSINESS BASIC ($200), ZTEL ($80) and TOP II ($100)
$730 Value **NOW $300**

COMBO

Includes: DEVELOPER II ($1280), ACCOUNTING PACKAGE ($300), QSAL ($200) and 6502X ($150)
$1930 Value **NOW $500**

LINKER - $80, A linking loader for handling the linkable modules created by the above assemblers.

DEBUG I - $80, A tool for debugging Z80 or 8080 code. Disassembles to CDL/TDL mnemonics compatible with above assemblers. Traces code even through ROM. Commands include Calculate, Display, Examine, Fill, Goto, List, Mode, Open File, Put, Set Wait, Trace, and Search.

DEBUG II - $100, A superset of Debug I. Adds Instruction Interpreter, Radix change, Set Trap/Conditional display, Trace options, and Zap FCB.

6502X - $150, A 6502 cross assembler. Runs on the Z80 but assembles 6502 instructions into 6502 object code! Similar features as our Macro assemblers.

QSAL - $200, A SUPER FAST Z80 assembler. Up to 10 times faster than conventional assemblers. Directly generates code into memory in one pass but also to offset for execution in its own memory space. Pascal like structures: repeat...until, if...then...else, while...do, begin...end, case...of. Multiple statements per line, special register handling expressions, long symbol names, auto and modular assembly, and more! This one uses ZILOG Mnemonics.

QED - $150, A screen editor which is both FAST and easy to learn. Commands include block delete, copy, and move to a named file or within text, repeat previous command, change, locate, find at start of line, and numerous cursor and window movement functions. Works with any CRT having clear screen, addressable cursor, clear to end of line, clear to end of screen, and 80X24.

ZTEL - $80, An extensive text editing language and editor modelled after DEC's TECO.

ZEDIT - $50, A mini text editor. Character/line oriented. Works well with hardcopy terminals and is easy to use. Includes macro command capability.

TOP I - $80, A Text Output Processor for formatting manuals, documents, etc. Interprets commands which are entered into the text by an editor. Commands include justify, page number, heading, subheading, centering, and more.

TOP II - $100, A superset of TOP I. Adds: embedded control characters in the file, page at a time printing, selected portion printing, include/merge files, form feed/CRLF option for paging, instant start up, and final page ejection.

ZDDT - $40, This is the disk version of our famous Zapple monitor. It will also load hex and relocatable files.

ZAPPLE SOURCE - $80, This is the source to the SMB ROM version of our famous Zapple monitor. It can be used to create your own custom version or as an example of the features of our assemblers. Must be assembled using one of our assemblers.

MODEM - A communication program for file transfer between systems or using a system as a terminal. Based on the user group version but modified to work with our SMB board or TRS-80 Models I or II. You must specify which version you want.

MODEM SOURCE - $40, For making your own custom version. Requires one of our Macro Assemblers.

DISASSEMBLER - $80, Does bulk disassembly of object files creating source files which can be assembled by one of our assemblers.

HARDWARE

S-100 — **SMB II Bare Board $50,** "System Monitor Board" for S-100 systems, 2 serial ports, 2 parallel ports, cassette interface, 4K memory (ROM, 2708 EPROM, 2114 RAM), and power on jump. When used with Zapple ROM below, it makes putting a S-100 system together a snap.
Zapple ROM $35, Properly initializes SMB I/II hardware, provides a powerful debug monitor.

IBM PC — **Big Blue Z80 board $595,** Add Z80 capability to your IBM Personal Computer. Runs CP/M programs but does not require CP/M or TPM. Complete with Z80 CPU, 64K add on memory, serial port, parallel port, time and date clock with battery backup, hard disk interface, and software to attach to PC DOS and transfer programs. Mfr'd by QCS.
50% Discount on all CDL software ordered at the same time as a Big Blue (and for the Big Blue).

APPLE II — **Chairman Z80 $345,** Add Z80 capability to your Apple II/II Plus computer. Runs CP/M programs with our more powerful TPM. Includes 64K memory add on (unlike the competition this is also useable by the 6502/DOS as well as the Z80), TPM, QSAL assembler, QED Screen Editor, and Business Basic. Mfr'd by AMT Research.
Apple Special $175, Buy the Apple Z80 Developer at the same time as the "Chairman" and pay only $175 instead of $325.

APPLE Z80 DEVELOPER

Includes: 6502X ($150), MACRO II ($100), MACRO III ($150), QSAL ($200), QED ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), ZDDT ($40) and BUSINESS BASIC ($200)
VALUE: $1250 **NOW $325**
$175 when purchased with AMT "Chairman" Board

DISK FORMATS

When ordering software specify which disk format you would like.

CODE	DESCRIPTION
8SD	8" IBM 3740 Single Density (128 bytes/26 sectors/77 tracks)
8DD	8" Double Density (256 bytes/26 sectors/77 tracks)
8XD	8" CDL Extended Density (1024 bytes/8 sector/77 traceks = 616K)
5SD	5.25" Single Density (TRS80 Model I, Versafloppy I, Tarbell I)
5EP	5.25" Epson Double Density
5PC	5.25" IBM PC Double Density
5XE	5.25" Xerox 820 Single Density
5OS	5.25" Osborne Single Density
5ZA	5.25" Z80 Apple (Softcard compatible)

TPM INFO When ordering TPM I or II, in addition to Disk Format, please specify one of the following codes:

	CODE	DESCRIPTION
TPM I:	NSSD/H	North Star Single Density for Horizon I/O
	NSSD/Z	North Star Single Density for Zapple I/O
	NSDD/H	North Star Double Density for Horizon I/O
	NSDD/Z	North Star Double Density for Zapple I/O
	TRS80-I	TRS-80 Model I (4200H Offset)
	TRS80II	TRS-80 Model II
	VI8	Versafloppy I 8"
	VI5	Versafloppy I 5.25"
TPM-II:	VII8	Versafloppy II 8" (XD)
	VII5	Versafloppy II 5.25"
	TRS80II	TRS-80 Model II (XD)

Prices and Specifications subject to change without notice.
TPM, Z80, CP/M, TRS80 are trademarks of CDL, Zilog, DRI and Tandy respectively.

ORDERING INFORMATION:

VISA/MasterCard/C.O.D.
Call or Write With Ordering Information....

OEMS:

Many CDL products are available for licensing to OEM's. Write to Carl Galletti with your requirements.

Dealer Inquiries Invited.

For Phone Orders ONLY Call Toll Free...
1-(800) 458-3491 (Except Pa.)
Ask For Extension #15

For information and Tech Queries call
(609) 599-2146

Computer Design Labs

342 Columbus Avenue/Trenton, NJ 08629

Circle no. 90 on reader service card.

RECLAIM (Text begins on page 14)

```
/*      RECLAIM.C

        A utility for crashed disk data reclaimation.

        By Walter V. Murphy   (415) 952-4484
        Compucations Computer Engineering
        212 Northwood Avenue
        South San Francisco, Ca. 94080

        RECLAIM is intended for non-commercial use and
        distribution.

This program accepts text input lines from STDIN containing
track, starting sector and filename information. The indicated
disk blocks are read and assembled into files on another disk
using the specified names. Many lines of input can be given to
the program by using input redirection.

The format of the input is as follows:
                Track,Sector,Filename.ext

Example:        64,16,PROGRAM.ASM
                65,48,DATAFILE.A01

When using directed input the command line looks like this:
                RECLAIM <INPUTFIL

Where INPUTFIL is a text file containing the input lines.
This allows generating a long sequence of disk blocks to
be read using a text editor.
*/

#define ERROR   -1      /* These declarations would usually be  */
#define TRUE    1       /* in a file called STDIO.H             */
#define FALSE   0
#define NULL    0
#define HOME    30
#define ESC     27
#define CR      13
#define LF      10
#define SECLEN  128     /* sector length */

/*
The external declaration is here because the Small-C this program
was intended for generates M80 assembly language source code.
*/
extern  stdin,stdout;

char *bufr;             /* block buffer */
int track,              /* for bios call */
    sector,             /* for bios call */
    blksiz,             /* number of sectors in a block */
    spt;                /* number of sectors per track  */
```

(Continued on page 20)

RECLAIM (Listing continued, text begins on page 14)

```
char filename[100],          /* holds the filename, s is the input line, */
     s[100];                 /* These don't need to be so long but they  */
                             /* are just for protection sake. */

main()
{    char *x,                /* x is a working input line pointer. */
          *addr;             /* addr is used for giving the DMA to cp/m. */
     int i,                  /* i is for general use. */
         fd;                 /* fd is a file descriptor number. */

     i=bdos(12,0);                   /* get cp/m version number */
     if (i/0x100) { signon();
         printf("\n  This program will not work with MP/M.",1); exit(1); }
     if ((i%0x100) < 0x20) { signon();
         printf("\n  This program will not work with less than CP/M 2.0.",
             1);
         exit(1); }

     if (!sysparms()) {signon(); /* get file system parameters */
                printf("\n  Unknown system error.",1); exit(1); }

                                     /* if memory is available- allocate it */
     if ((bufr= malloc(SECLEN*blksiz))==NULL) { signon();
                printf("\n  Not enough memory for buffer.",1); exit(1); }
```

```
    clrscreen();
    signon();
    printf("\n   This disk has %d sectors per track ",spt,2);
            printf("and %d sectors per block.\n",blksiz,2);

    gets(s);                        /* get line from stdin */
    while  (*s!=NULL) {
        x=s;
        track=atoi(x);              /* first number in line is track */
        while (isdigit(*x)==TRUE) ++x;
        while (isdigit(*x)==FALSE) ++x;
        sector=atoi(x);             /* second number in line is sector */
        while (isdigit(*x)==TRUE) ++x;
        while (isalpha(*x)==FALSE) ++x;
        strcpy(filename,x);         /* the filename is last */

/* If you want the second field in your input line to be a track
   relative block number rather than sector number then include
   the following line:
        sector= ((sector-1)* blksiz) +1;
*/
        printf("  Track: %d, Starting Sector: %d, Filename: %s\n",
                track,sector,filename,4);
        if (sector > spt) {
                printf("  The sector requested is too big.",1);
                exit(1);
                }
```

(Continued on next page)

RECLAIM (Listing continued, text begins on page 14)

```
        gets(s);                    /* get line from stdin */
        i=0;  addr= bufr;           /* now fill the block buffer */
        while ((dread(track,sector,addr) != ERROR) && (i < blksiz)) {
                addr+=SECLEN; i++;
                if ((sector+=1)>spt) {sector=1; track+=1;}
                }

                    /* blksiz is number of sectors to write */
        if ((fd=creat(filename,0))==ERROR)
           {printf("  Unable to create that file for you.",1); exit(1);}
           else i=write(fd,bufr,blksiz);

        if ((i==ERROR) || (i!=blksiz))
                {printf("  Error while writing that file.",1); exit(1);}
        close(fd);
    }
    printf("\7  Successful completion. Goodbye from RECLAIM.",1);
}
                /* End of main procedure */


signon()
{
printf("  COMPUCATIONS Disk block RECLAIMation utility, 12Aug82.",1);
}

/* Procedure to read a sector using the BIOS. */
/* Returns true or false for success or failure. */

dread(trk,sec,addr)
    int trk,sec;
    char *addr;
{   bios(10,trk);                   /* select the track */
    bios(11,sec);                   /* select the sector */
    bios(12,addr);                  /* give DMA address */
    if (bios(13,1)) return 0;       /* perform the read */
    return 1;                       }


/* Procedure to write a sector using the BIOS. */
/* Returns true or false for success or failure. */

dwrite(trk,sec,addr)
    int trk,sec;
    char *addr;
{   bios(10,trk);                   /* select the track */
    bios(11,sec);                   /* select the sector */
    bios(12,addr);                  /* give DMA address */
    if (bios(14,1)) return 0;       /* perform the write */
    return 1;                       }


/* get system parameters from the BIOS. Set values for
   sector count and number of sectors in an allocation block.
   return true if ok.   */
```

```
sysparms()
{    int *dph,i;
     char *dpb,*bsh;
     i=bdos(25,0);
     if ((dph=bios(9,i))==0) return FALSE;
     dpb=dph[5];                      /* get disk parameter block */
     spt=(dpb[1]*0x100)+dpb[0];       /* get # sectors per track   */
     bsh=dpb[2];                      /* get block shift factor    */
     blksiz=1;                        /* 2**bsh is # sectors per block */
     while (bsh) {blksiz=blksiz*2; bsh--;}
     return TRUE;
}

/* carriage return, line feed
*/
nl()
{    bdos(2,CR); bdos(2,LF);
}

/* clear screen
*/
clrscreen()
{    bdos(2,HOME); bdos(2,ESC); bdos(2,'Y');
}

/* Interface to call the bios.
*/
bios(fun,arg)
     int fun,arg;
{    char *ofs;
     ofs=(fun-1)*3;
#asm
         pop      d         ;offset
         pop      h         ;ret
         pop      b         ;arg
         push     b
         push     h
         push     d
         lhld     1         ;pointer to bios
         dad      d         ;add offset
         call     gohl      ;call indirect thru HL.
         xchg
         mov      l,a
         mvi      h,0
         pop      b         ;ofs
         push     b
         mov      a,c
         cpi      (9-1)*3 ;select disk function ?
         jnz      biosout
         xchg               ;if yes put the dph addr in DE.
biosout:
#endasm
}

#asm
gohl:    pchl
#endasm
```

End Listing

Binary Magic Numbers
Some Applications and Algorithms

This article presents a set of constants that enhances many different kinds of binary calculations. These "magic numbers" can be used to determine the positions of bits within words; to reverse, permute, and map bits within words; to compute sideways sums and parity; and to convert Gray code values to their binary equivalents. An understanding of these numbers and their applications can benefit the machine language programmer and can improve code generation in compilers. The applications for these constants will arise naturally in the construction of algorithms for the solution of each problem.

Introduction

The field of mathematics has uncovered many "magic numbers." These are numbers which have special or unusual properties when used in certain calculations. For example, the magic number 142,857 is the smallest "cyclic number."[1] Cyclic numbers produce a result that is a cyclic permutation of the original number's digits when multiplied by an integer less than or equal to the number of digits in the original number:

1 * 142,857 = 142,857
2 * 142,857 = 285,714
3 * 142,857 = 428,571
4 * 142,857 = 571,428
5 * 142,857 = 714,285
6 * 142,857 = 857,142

Most magic numbers have little or no application apart from being mathematical curiosities, and no computer uses are known for them. Cyclic numbers are no exception to this; they are elegant but have no practical value. One of the reasons for this is that many numerical tricks like cyclic numbers rely on decimal (base 10) format. Most computers operate uncomfortably at best using base 10 arithmetic and prefer to stick to binary (base 2). Most exploration for number "magic" has been conducted in the decimal realm, so there has been only limited discussion of binary magic numbers.

Binary magic numbers do exist, and these numbers can be used to significantly enhance certain computer operations.

by Edwin E. Freed

Edwin E. Freed, Mathematics Department, Harvey Mudd College, Claremont, California.

This article concentrates on a specific sequence of binary magic numbers. The Nth number in this sequence will consist of an infinite sequence from right to left of 2^N 1's, followed by 2^N 0's, followed by 2^N 1's, and so on:

...0101010101010101
...0011001100110011
...0000111100001111
...0000000011111111
⋮

In practice, however, binary operations are limited to a finite word size. For a word size of 8 bits, there will be three numbers:

B[1] := 01010101
B[2] := 00110011
B[3] := 00001111

There will be four B-constants for word sizes of 9-16 bits, five for sizes of 17-32 bits, and so on. The bit complements of these binary values will also prove to be useful:

B[4] := 10101010
B[5] := 11001100
B[6] := 11110000

Notation

All of the example algorithms in this article are presented in Pascal. The reader is assumed to be familiar with Pascal and with the basics of boolean algebra.

Pascal does lack full-word logical operations (logical functions which apply on a bit-for-bit basis to each bit in an integer value) as standard functions, so the following special functions are assumed to be implemented:

IAnd (X, Y)	– return bit-for-bit AND of X and Y.
IOr (X, Y)	– return bitwise inclusive OR of X and Y.
IXor (X, Y)	– return bitwise exclusive OR of X and Y.
INot (X)	– return bit-for-bit complement of X.
LShift (X, N)	– return X shifted left by N bit positions. Zeroes are shifted into the low-order bits.
RShift (X, N)	– return X shifted right by N bit positions. Zeroes are shifted into high-order bits.

Some of the methods shown assume integer overflows will not be detected. No other extensions to Pascal are used. All variables are explicitly declared and the integer type is the only type of variable used. Some global machine-dependent constants are assumed to have been declared outside the example routines. N is the number of bit positions in an integer value. V is the base 2 logarithm of N rounded to the next highest integer. The values B[1] through B[2*V] are considered to be global constants and are frequently used in the routines. B[1] through B[V] will be the binary magic numbers defined above and B[V+1] through B[2*V] will be their bit-for-bit complements. The constants S[1] through S[N] are a table of the powers of 2 within the range of the machine. S[Q] will be equal to 2^Q. The global constants might be initialized on a 12-bit machine by th Pascal code fragment in Figure 1, below.

Most of the algorithms will be more useful if they are recoded in a specific machine language. No sophisticated Pascal constructs like sets or records are used, so the translation process should be quite

```
const
  N = 12; V = 4;
var
  B : array [1..8] of integer;
  S : array [1..12] of integer;
  I : integer;
begin
  B[1] := 010101010101; B[2] := 001100110011;
  B[3] := 111100001111; B[4] := 000011111111;
  B[5] := 101010101010; B[6] := 110011001100;
  B[7] := 000011110000; B[8] := 111100000000;
  for I := 1 to N do S[I] := LShift (1, pred (I));
  ...
```

Figure 1.

easy. The special logical functions will correspond to single instructions on most machines.

Some of the algorithms discussed below will involve the positions of bits within words. Some consistent numbering scheme for these bits is required. The least significant bit of a value is called bit 0 and numbering proceeds to the left. The sign bit of the value (most significant bit) will be bit (N-1). If the eight digit positions in an 8-bit word were each filled with their corresponding numbers, the result would be 76543210. Note that this numbering scheme can be derived from the constants B[V+1] through B[2*V] if they are read as binary values from bottom to top in each digit column. This is a very important property of these numbers that will be used frequently.

A few of the algorithms assume that negative values are stored in two's-complement form. This is usually not a problem – there are not very many one's-complement or signed-magnitude machines.

Locating Bits

One commonly needed binary operation is the ability to locate the lowest or highest zero or nonzero bit in a value. This is often used in operating systems, when sets of boolean flags are grouped together in a single word and each flag requires a specific action. Bit location operations are of particular interest in scheduling algorithms, where quick execution is essential.

Many large machines include special "exotic" instructions to locate zero or nonzero bits in a value. The DEC-10 has the JFFO instruction, which is a mnemonic for "jump if found first one."[2] This operation tries to find the highest nonzero bit in a 36-bit value and optionally branches if the operation is successful. By contrast, the VAX-11 instructions set includes the FFS and FFC instructions, which are mnemonics for "find first set" and "find first clear," respectively.[3] These instructions search a field (an extracted portion of a 32-bit value) from the lowest bit up for the first zero or nonzero bit. Neither machine includes an explicit instruction for searching in the opposite direction, although the CVTLD (convert long integer to double-precision floating) instruction on the VAX-11 might be used for this. It is also interesting to note that the instruction execution speed is bit-position dependent on the DEC-10 (the farther away from an 18-bit half-word boundary, the longer the instruction takes) and is position independent on the VAX-11 (the same amount of time is taken for each bit position).[4] It is vital to be aware of such speed dependencies if exotic instructions like these are used.

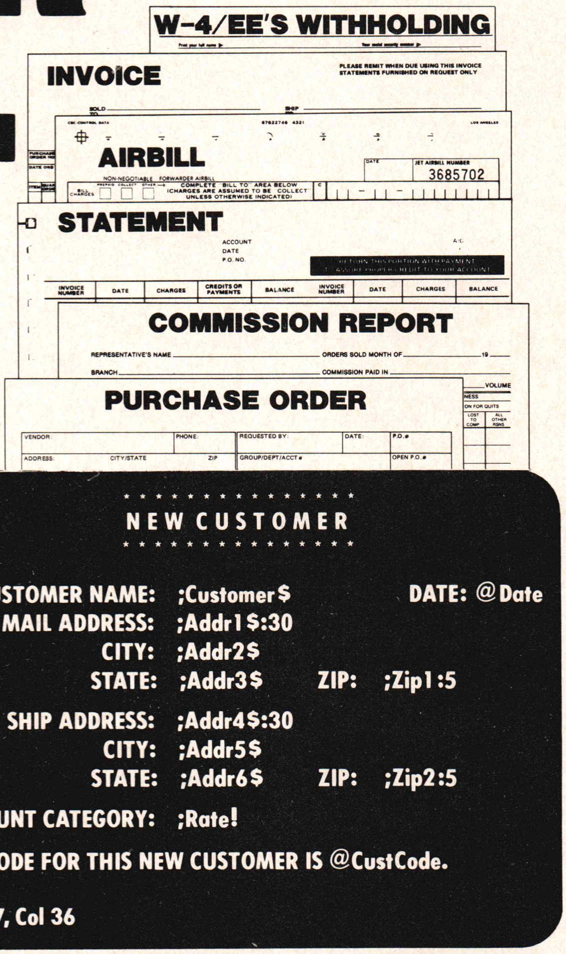
Smaller machines do not usually include such special instructions. The most obvious way to perform this operation is to successively shift the value in question until a bit in the proper state is found. The lowest bit can be checked with an AND, and the highest bit can be checked by watching for a sign change. The functions in Figure 2, page 28, illustrate these methods.

These functions return a result of -1 if no bit that satisfies the specified condition can be found, and the bit position of the selected bit otherwise. Up to N iterations within the functions may be necessary to find the selected bit. However, it is possible to make use of the B-constants and check more than one bit position with each iteration. The functions in Figure 3, page 28, perform complementary operations to the functions shown above by making use of the B-constants. Note that the LowClearBit function operates on the complement of the input value and not on the value itself.

The functions operate on multiple bit fields at once. Each iteration of the **for** loop isolates the desired bit to one-half of the remaining bits in a word. The weighting of the bit positions is used to advantage in computing the final position by shifts and adds instead of using a table of constants like S. The S-constants could be used if the shift operation is difficult to implement. The assignment of Result within the **for** loop is superfluous during the first iteration and may be skipped.

These algorithms take V iterations in all cases. This can be far less than N for

END THE PAPER CHASE.



Now you can computerize your business forms and input screens without all that tedious, time consuming programming.

With ZIP,™ it's going to seem almost too easy.

Just "paint" the screen.

To prepare an input screen or output form, just move the cursor around the screen and type in text, prompts and data field names exactly where you want them. Use "@" to print or display values, use ";" for input fields.

When the screen looks like the format you want, type "/S" and what you see is what you'll get. In seconds, ZIP writes all the program code to recreate the format on the screen or on paper.

And you can use the ZIP code immediately just by adding a single line to your main program: GOSUB xxxxx in BASIC, DO Formname in dBASE II.

ZIP is quick and friendly.

ZIP runs on virtually every 8-bit micro known to man, and your terminal continues to work the way it did (tab, arrows, etc.), so you don't have to learn or unlearn anything about your equipment.

Commands are typed (no control codes), so you or your secretary can breeze through input screens and output forms up to 88 lines long and have ZIP whip out the BASIC or dBASE II code.

The ZIP Talker,™ a line at the bottom of the screen, always tells you exactly where you are. And Help is just two keystrokes away.

Now MBASIC really ZIPs.

The MBASIC version goes further and gives your programs the same screen handling characteristics that ZIP has, by writing a piece of itself in MBASIC so that you can use it in any of your programs.

The operator can use the arrows, etc. during data entry and conveniently jump back and forth between the input fields.

You can specify field lengths, or let ZIP default to the available space. Either way, text and prompts are protected no matter what kind of terminal you have, so the operator can't write over the fields and prompts.

You get the MBASIC code for a "Talker" that you can use to pretty up your program prompts.

And easy, one-line data validation is built in.

Join thousands of users ZIPping along.

All you need is an 8-bit micro with CP/M or MPM, 48k of memory and a 24x80 ASCII or ANSI terminal (Osborne 1 and 56k Apple okay, too).

The MBASIC and CBASIC versions are $160 each ($225 for both) plus $7 shipping (VISA, MasterCharge or money order). The dBASE II version is available alone from Ashton-Tate (213-204-5570), or we'll sell you dBASE II with ZIP for $650. For more information, contact Nexus, 5455 Wilshire, Suite 802, Los Angeles, CA 90036.

Or if you'd like to end the paper chase sooner, just call 800-227-3747. (In California, call 213-937-0554, add 6% tax.)

ZIP™ from **Nexus**
The man-machine connection.

large word sizes, but the iterative loop operation is more complex. The distribution of bits in the input values that appear in a given application must also be considered. The LowSetBit function presented in Figure 2 may use an average of fewer than V iterations if nonzero low-order bits are prevalent in the input values.

It is possible to simplify the last two functions if it is known that there is only a single nonzero bit in the value. It becomes unnecessary to replace the input value as the algorithm proceeds since the desired bit is already isolated, so the Temp and Ival variables are not needed.

It is very easy to check for a single nonzero bit in a word. Given Value, "((Value <> 0) **and** (IAnd (Value, pred (Value)) = 0))" will be true if and only if there is a single nonzero bit in Value.[5] An alternate test is "((Value <> 0) **and** (IAnd (Value, -Value) = Value))."

Sideways Addition

The sideways sum of a number is the sum of all the individual digits in the number taken one at a time. The sideways sum will be different depending on the base the number is represented in. For example, the sideways sum of the decimal value 12345 is 15, and the sideways sum of the binary value 10110 is 11 (in binary). A binary sideways sum is just the count of the 1's in the binary value.

Sideways sums have many interesting mathematical properties and applications. Decimal sideways sums are the basis for an old trick for determining divisibility by 3 or 9. If a number is evenly divisible by 3 then the decimal sideways sum will also be evenly divisible by 3. The same property holds true for 9 as well. (The decimal sideways sum always gives a value with the same remainder modulo 3 or 9 as the original value.) Since the sideways sum of a number is usually much smaller than the number itself, it is easy to check divisibility of a large number by iteratively computing sideways sums until a manageably small value is obtained.

A binary value can be thought of as a representation of a set. Each digit corresponds to a single element that may possibly be in the set. A digit value of 0 means the element is not present and a value of 1 means the element is present. In this scheme (which is used by many programming languages), the binary sideways sum indicates the cardinality of a set – it is a count of the number of elements in the set.

Some machines have implemented the binary sideways sum as an instruction. The CDC (Control Data Corporation) 6400, 6500, and 6600 series processors implement the "CXi Xk" instruction. This instruction counts the number of nonzero bits in register Xk and stores the result in register Xi. i and k are single

```
function LowSetBit (Value : integer) : integer;
var
  Temp, Result : integer;
begin
  if Value = 0 then LowSetBit := -1
  else begin
    Temp := Value;
    Result := 0;
    while not odd (Temp) do begin
      Temp := RShift (Temp, 1);
      Result := succ (Result);
    end;
    LowSetBit := Result;
  end;
end;

function HiClearBit (Value : integer) : integer;
var
  Temp, Result : integer;
begin
  Temp := Value;
  Result := pred (N);
  while Temp < 0 do begin
    Temp := LShift (Temp, 1);
    Result := pred (Result);
  end;
  HiClearBit := Result;
end;
```

Figure 2.

```
function LowClearBit (Value : integer) : integer;
var
  Temp, Result, Ival, I : integer;
begin
  if Value = -1 then LowClearBit := -1
  else begin
    Ival := INot (Value);
    Result := 0;
    for I := V downto 1 do begin
      Result := LShift (Result, 1);
      Temp := IAnd (Ival, B[I]);
      if Temp = 0 then Result := succ (Result)
      else Ival := Temp;
    end;
    LowClearBit := Result;
  end;
end;

function HiSetBit (Value : integer) : integer;
var
  Temp, Result, Ival, I : integer;
begin
  if Value = 0 then HiSetBit := -1
  else begin
    Ival := Value;
    Result := 0;
    for I := V downto 1 do begin
      Result := LShift (Result, 1);
      Temp := IAnd (Ival, B[V+I]);
      if Temp <> 0 then begin
        Result := succ (Result);
        Ival := Temp;
      end;
    end;
    HiSetBit := Result;
  end;
end;
```

Figure 3.

```
function SideSum (Value : integer) : integer;
var
  Temp, Result : integer;
begin
  Temp := Value;
  Result := 0;
  while Temp <> 0 do begin
    Result := Result + IAnd (Temp, 1);
    Temp := RShift (Temp, 1);
  end;
  SideSum := Result;
end;
```

Figure 4.

```
function SideSum (Value : integer) : integer;
var
  Temp, Result : integer;
begin
  Temp := Value;
  Result := 0;
  while Temp <> 0 do begin
    Result := succ (Result);
    Temp := IAnd (Temp, pred (Temp));
  end;
  SideSum := Result;
end;
```

Figure 5.

```
function SideSum (Value : integer) : integer;
var
  Result, I : integer;
begin
  Result := Value;
  for I := 1 to V do
    Result := IAnd (RShift (Result, S[I]), B[I]) +
      IAnd (Result, B[I]);
  Sidesum := Result;
end;
```

Figure 6.

```
function Parity (Value : integer) : integer;
var
  Temp : integer;
begin
  Temp := Value;
  Result := 0;
  while Temp <> 0 do begin
    Result := IXor (Temp, Result);
    Temp := RShift (Temp, 1);
  end;
  Parity := IAnd (Result, 1);
end;
```

Figure 7.

octal digits and serve to identify one of the eight 60-bit X registers available on the machine. The time used for this operation varies even within this series of machines; the 6400 and 6500 require 68 minor cycles while the 6600 needs only eight. For comparison purposes, an integer addition takes six minor cycles on the 6400 and 6500 and three minor cycles on the 6600.[6]

The most obvious way to compute a binary sideways sum is to iteratively shift the value right and count the low-order bits as shown in Figure 4, at left.

This method can take up to N iterations, but it typically takes fewer than N since small values are fairly common. There are many possible variations on this basic type of function, including using bits shifted out the top of a value, shifting an entire field out of the value and looking up its sideways sum in an auxiliary table, and using floating-point normalization hardware if it is available.

The test for a single nonzero bit presented above may also be adapted to produce a sideways summation algorithm (see Figure 5, at left). The number of iterations for this method is equal to the final sideways sum result. The average number of iterations would be N/2 if a random value (each bit has an equal chance of being 0 or 1) is input.

It is possible, however, to operate on multiple fields within a value simultaneously, first adding adjacent bits, then bit pairs, and so on. The coding of this method is set forth in Figure 6, at left. This method uses only the uncomplemented B-constants and takes V iterations in all cases. There is only a slight increase in the number of operations required in the loop over the first SideSum function. This method is preferred when a large number of 1's are expected in the input value.

Parity

A slight variation on sideways addition is to just record whether or not the sum of the digits is even or odd. This result is called the parity of a number and applications for it frequently arise in error correction techniques.

Parity is usually thought of as the exclusive-OR of all the binary digits in a value – thus a 0 indicates even parity and a 1 indicates odd parity. Many microcomputers include hardware to compute parity bits easily, but many larger machines do not have such facilities. The simplest way to compute parity, illustrated in Figure 7, at left, is analogous to the first method for sideways sum computation involving repetitive shifts and tests.

As before, this operation can take up to N iterations but typically requires somewhat fewer. The method can be altered to take advantage of link bits or floating-point normalization instructions

```
function Parity (Value : integer) : integer;
var
  Result, I : integer;
begin
  Result := Value;
  for I := 1 to V do
    Result := IXor (Result, RShift (Result, S[I]));
  Parity := IAnd (Result, 1);
end;
```

Figure 8.

```
function IXor (X, Y : integer) : integer;
begin
  IXor := IAnd (INot (IAnd (X, Y)), IOr (X, Y));
end;
```

Figure 9.

```
function Parity (Value : integer) : integer;
var
  Result, I : integer;
begin
  Result := IAnd (IAnd (RShift (Value, 1), B[1]) +
    IAnd (Value, B[1]), B[1]);
  for I := 2 to V do
    Result := IAnd (RShift (Result, S[I])
      + Result, B[1]);
  Parity := IAnd (Result, 1);
end;
```

Figure 10.

(which typically shift the uppermost nonzero bit to the highest bit position automatically). For example, the 8087 numeric data processor can be used in this fashion by reading an integer value into the unit and subsequently writing the value out in floating-point form.[7] The mantissa of the result will then be the original integer shifted left so that the highest nonzero bit is at the top of the field, and the exponent is an indication of the number of shifts that were needed. Care must be taken to intercept negative values and to use a floating-point format capable of holding an entire integer in its mantissa without loss of accuracy. The 8087 is slightly cumbersome to use in this fashion as it requires reading and writing the values into memory instead of performing the operation entirely in registers. Machines with simple conversion instructions are actually more versatile in this case.

The second method presented for sideways addition can also be adapted to compute parity. This algorithm is easily derived from the SideSum function and will not be presented here.

Figure 8, at left, shows how the third method for sideways addition can be adapted into a parity function with considerable ease. This algorithm always takes V iterations. The B-constants are not required since exclusive-OR operations do not generate carries and masking is not necessary.

One peculiarity that must occasionally be dealt with is machines that do not possess exclusive-OR hardware. Of course, the exclusive-OR may be coded in terms of other logical operations (see Figure 9, at left). This is a very costly operation. It may be more practical to compute the full sideways sum and extract the lowest bit rather than using this cumbersome function.

The third sideways sum technique may alternately be adapted to provide the parity of a value using only ANDs, shifts, and normal addition. This is accomplished by noting that an exclusive-OR is essentially an addition with no carry, so all that must be done is to prevent additive carries from crossing from one field of the number to the next, as in Figure 10, at left. This technique is interesting in that it requires only one of the binary magic numbers. (V-1) iterations are required, but some preliminary calculations of greater complexity than the iterative one must be performed.

Another peculiarity of some machines is the lack of an AND instruction. Such machines, notably the LSI-11, PDP-11, and VAX-11, have a bit test instruction (BIT) that performs an AND but does not store the result. These machines also have a bit clear instruction (BIC), which corresponds to an "AND-NOT" operation.[3] This does not present a problem with any of the algorithms shown in this

Circle **no. 75** on reader service card.

article. Most uses of IAnd involve a constant. The constant is just complemented and then the BIC operation can be used. One exception to this is the IXor function, which can be recoded more simply using the statement "IXor := IOr (BIC (X, Y), BIC (Y, X))." The only other exceptions are the tests for a single nonzero bit. The first test does not require the storage of the result, so a BIT instruction could be used or an INot can be inserted. The second test can be expressed as "((Value <> 0) **and** (BIC (Value, pred (Value)) = Value))."

Weighted Sideways Addition

Sometimes it is necessary to perform sideways addition with each digit weighted differently. This requires multiplying each digit by a fixed constant before summation is done. The way numbers are normally interpreted is merely a special case of this operation; each digit of a number is weighted by successive powers of the base of the number system. Thus numbers themselves can be thought of as just collections of digits where a particular type of weighted sideways addition is used to compute their value. The conventional sideways sum is another special case – each digit has unary weight.

Weighted sideways sums can be used to generate arbitrary mappings and permutations of bits within words. A permutation is a rearrangement of the bits within a word where no bits are lost, and a mapping is a more general operation in that a bit may be placed in more than one position in the final result and certain bits may be lost. It is trivial to construct a set of weights for a sideways sum that will accomplish any mapping or permutation – each bit's weight is a mask containing 1's in the bit positions the bit is to be mapped into.

The additive properties of a weighted sideways sum can be used in a more general sense, however. Consider a binary value as a representation of a set, as it has been discussed earlier. Each potential element of the set might be said to have certain properties. Different elements will have different properties. There can be many properties associated with each member. Take the set of fruits [LIME, LEMON, ORANGE]. One property might be color – [GREEN, YELLOW, ORANGE]. Another might be cost – [41 cents, 35 cents, 52 cents]. A third might be ordinal position – [first, second, third].

Some properties are not additive (like color) and others (position or cost) are. It may not make sense to total the positions of the elements of a set, but a property like cost might be useful to sum over an entire set. In the binary representation of a set, the costs could be thought of as weights and the resultant sideways sum as the cost of the entire set.

The straightforward way to compute a weighted sideways sum is to maintain a table of constant weights and to iteratively shift the input value. For the function shown in Figure 11, page 33, a table of constant weights C[1] through C[N] applying to each respective bit position is assumed to be defined.

This routine takes a maximum of N iterations, which is to be expected. There is no clear way to write a more efficient routine using the B-constants. It is necessary to examine the constant weights in C for certain bit patterns. Consider the following ascending sequence for an 8-bit machine:

```
C[1] := 0001        C[5] := 0101
C[2] := 0010        C[6] := 0110
C[3] := 0011        C[7] := 0111
C[4] := 0100        C[8] := 1000
```

A new set of weights can be derived by reading the columns of the C array. This is equivalent to constructing an 8 x 8

matrix with the rows of C and taking the reversed rows of the transpose of this matrix. This will create a new set of constants D[1] through D[K]. K = 4 for the example C-constants given above, since D[5] through D[8] will all be zero. In this example, the D-constants are actually shifted versions of the B-constants.

Now the D-constants can be used to reduce the weighted summation to a series of unweighted sideways sums. Each unweighted sum builds a single quantity of constant weight for the final result, as illustrated in Figure 12, below.

This method is by no means faster than the conventional version of WeightedSum for a general set of weights. However, certain special cases (such as the example weights given in Figure 12) will introduce a multitude of simplifying reductions if the SideSum function is coded inline, particularly if the D-constants bear some resemblance to the B-constants. The reduction of the example is left as an exercise to the reader. Which WeightedSum method takes fewer operations?

Bit Reversal

Another interesting operation to study is that of reversing all the bits in a word. Bit 0 will become bit (N-1), bit 1 will become bit (N-2), bit 2 will become bit (N-3), and so on. This is actually a special case of the weighted sideways sum operation, since it is nothing more than a permutation of the bit positions in a value. One application for bit reversal has already been encountered in the construction of the D array from the C-constants for the second method of weighted sideways addition.

Bit reversal may easily be implemented by a series of shifts (see Figure 13, below). N iterations are always required for this function.

This function may be modified to

```
function WeightedSum (Value : integer) : integer;
var
  Temp, Result, I : integer;
begin
  Result := 0;
  Temp := Value;
  I := 1;
  while Temp <> 0 do begin
    if odd (Temp) then Result := Result + C[I];
    Temp := RShift (Temp, 1);
    I := succ (I);
  end;
  WeightedSum := Result;
end;
```

Figure 11.

```
function WeightedSum (Value : integer) : integer;
var
  Temp, Result, I : integer;
begin
  Result := 0;
  for I := 1 to K do begin
    SideSum (IAnd (Value, D[I]), Temp);
    Result := LShift (Result, 1) + Temp;
  end;
  WeightedSum := Result;
end;
```

Figure 12.

```
function ReverseBits (Value : integer) : integer;
var
  Temp, Result, I : integer;
begin
  Temp := Value;
  for I := 1 to N do begin
    Result := LShift (Result, 1) + IAnd (Temp, 1);
    Temp := RShift (Temp, 1);
  end;
  ReverseBits := Result;
end;
```

Figure 13.

```
function ReverseBits (Value : integer) : integer;
var
  Result, I : integer;
begin
  Result := Value;
  for I := 1 to V do
    Result := IAnd (LShift (Result, S[I]), B[V+I]) +
       IAnd (RShift (Result, S[I]), B[I]);
  ReverseBits := Result;
end;
```

Figure 14.

```
function GrayToBinary (Value : integer) : integer;
var
  Temp, Result : integer;
begin
  Result := 0;
  Temp := Value;
  while Temp <> 0 do begin
    Result := IXor (Result, Temp);
    Temp := RShift (Temp, 1);
  end;
  GrayToBinary := Result;
end;
```

Figure 15.

reverse the bits in a quantity (2*N) bits long. The high-order part of the input quantity is passed in Result, and all that is necessary is to modify the function to copy the sign bit of Result to the sign bit of Temp on each iteration. The final value will be contained in Temp and Result. No additional iterations are needed.

Many small computers have an additional bit which is attached to the high-order end of the register used for shift operations. This bit is usually called the link or carry flag. This bit can be a nuisance when performing some circular rotate operations since it acts as an extra bit position between the least significant bit and the sign bit. Nevertheless, a carry flag can be used to advantage in bit-reversal routines. For example, the Motorola 6800 can be used for bit reversal of a 16-bit quantity if the high-order 8 bits of the input value are placed in accumulator A, the low-order 8 bits of the input value are placed in accumulator B, the following two instructions are repeated eight times:

```
ROL A
ROR B
```

and finally a single "ROL A" instruction is executed. The result will be in the accumulators in the same configuration as the input value. The H, I, and C flags will be unaffected and the N, Z, and V flags will be set according to the high-order 8 bits of the result.

It is possible to use the B-constants to minimize the number of iterations required as in Figure 14, at left. This method is restricted to machines where N is a power of 2. This will not be a problem on most processors since word lengths of 8, 16, and 32 bits are the most common. There are some processors, however, where this method cannot be implemented, like the PDP-8 and Intersil 6100 (12-bit), the DEC-10 (36-bit), and most CDC machines (60-bit). This method requires only V iterations in all cases. It should be avoided on machines which have difficulty in shifting a quantity by more than one bit position at once, and on machines whose word length is not a power of 2.

Gray Codes

Gray codes are essentially an alternate set of counting numbers that have different rules than most conventional number systems. When Gray code numbers are placed in counting order, any adjacent pair of numbers will differ in at most one digit position, and the absolute value of the difference in that digit position will always be one.

There are an infinite number of Gray codes, and Gray codes exist for every numeric base. The simplest Gray code of all is the binary reflected Gray code, which will be the code under discussion here. The binary reflected Gray code is used more often than any other Gray code.

Gray codes have many applications such as error correction in pulse-code modulated systems. There are several places in recreational mathematics where Gray codes turn up; the disk motion of the Towers of Hanoi problem is an example.[8]

It is easy to convert an ordinary binary number into its binary reflected Gray code equivalent. If Value is the binary number, "IXor (Value, RShift (Value, 1))" will be the Gray code equivalent for Value. The conversion of a Gray code number to its binary equivalent is not so easy. Each bit of the original binary number may be recreated by taking the exclusive-OR of the corresponding bit in the Gray code with all the bits to its left (higher-order bits including the sign bit). For example, a binary value of 0111 would be converted to 0100 in Gray code.

The conversion of a Gray code value into its original binary form may be accomplished by the function in Figure 15, page 34. The function requires at most N iterations to complete the operation.

The final value created can be represented in tabular form as an "addition problem," where exclusive-OR is used instead of normal addition. If each bit position in Value is labeled with a letter "a" through "n" (assuming that N=8 and "a" is the label for the highest bit), the problem can be seen as:

```
abcdefgh
 abcdefg
  abcdef
   abcde
    abcd
     abc
      ab
       a
--------
 Result
```

In order to solve the problem in V iterations, it is useful to first define the set of augmented constants T[1] through T[V] by the relationship "T[Q] := IAnd (LShift (B[Q], 1), LShift (B[Q], S[Q]))." The T-constants are shown here for N = 16:

```
T[1] := 1010101010101010
T[2] := 0100010001000100
T[3] := 0001000000010000
T[4] := 0000000100000000
```

The Nth T-constant is formed by repeating a pattern from right to left of (2^{N-1}) 0s, followed by a single 1, followed by $(2^{N-1}-1)$ 0s.

The method used with the T-constants is vaguely similar to the "divide and conquer" method of Ercegovac.[9, 10] Each iteration builds several different fields of the final result. Multiplication is used in the loop – this method should be avoided on machines which do not have integer multiplication hardware. The multiply operation is used to shift single-bit fields

```
function GrayToBinary (Value : integer) : integer;
var
  Result, I : integer;
begin
  Result := Value;
  for I := 1 to V do
    Result := IXor (Result, RShift (IAnd (Result,
        T[I]), S[I]) *
        (pred (S[succ (S[I])]));
  GrayToBinary := Result;
end;
```

Figure 16.

into multiple positions in the word (see Figure 16, page 35).

The pattern generated by this function is quite interesting. After the first iteration, the value in Result is (using the same "addition problem" notation as above and assuming N = 8):

```
abcdefgh
 a c e h
--------
 Result
```

After the second iteration, the pattern is:

```
abcdefgh
 abc efg
  ab  ef
   a   e
--------
 Result
```

On the third and final iteration, the binary equivalent is obtained.

Generation of Constants

It may be necessary in some programming environments to create the B-constants dynamically instead of just reading them from a table. The B-constants are easily generated using the S-constants (which are in turn quite easy to create using a single LShift). See Figure 17, at right.

The dynamic generation of the B-constants may be "blended" with the

```
procedure CreateBConstants;
var
  Temp, I : integer;
begin
  Temp := -1;
  for I := V downto 1 do begin
    Temp := IXor (Temp, LShift (Temp, S[I]));
    B[I] := Temp;
    B[V+I] := INot (Temp);
  end;
end;
```

Figure 17.

```
procedure CreateTConstants;
var
  Temp, I : integer;
begin
  Temp := 1;
  for I := V downto 1 do begin
    T[I] := LShift (Temp, S[I]);
    Temp := IOr (Temp, T[I]);
  end;
end;
```

Figure 18.

inner loop of most of the algorithms given in this article. Each iteration of the loop then generates the constant it requires using a temporary variable that is initially set to -1. Unfortunately, it does not appear to be possible to perform this operation in the reverse order, that is, derive B[Q+1] from B[Q]. This makes it difficult to add the code for dynamic generation of the B-constants to the inner loop of the SideSum function. Although the ReverseBits function loop is written in ascending order, it may operate in either direction, so it is possible to generate the B-constants as the loop proceeds.

The T-constants used in the GrayToBinary algorithm may be generated from the B-constants by using their defining relationship. Alternately, a similar iterative process can be used, which also proceeds in the wrong direction to make it useful within the inner loop of the GrayToBinary function (see Figure 18, page 36). The assignment of Temp in the last iteration of the loop is not needed and always sets Temp to -1.

Advanced Applications

More sophisticated applications for the B-constants exist. They can be used to transpose bit matrices[11, 12] and there are even applications for these constants in certain error correcting codes. These applications are beyond the scope of this article, and might be the subject of a future one.

Conclusion

This article has covered many applications for this particular set of binary constants. The behavior of all these algorithms seems to have a lot of things in common – the normal and obvious method for performing the operation typically takes up to N iterations, where N is the word size of the machine, and the method using the B-constants takes V operations, where V is the base 2 logarithm of N rounded up to the nearest integer. For large word sizes (or when operating on multiword values with a smaller machine), these algorithms represent a considerable savings in execution time. Judicious use of these techniques may improve many machine-language programs.

Acknowledgements

I wish to thank Donald Knuth, whose work provided the basis of many of the ideas expressed in this article.[11] I also wish to thank Douglas Grover and Kevin Carosso of the MATHLIB project, who provided the problems to solve that led to this article along with many helpful suggestions.

References

1 M. Gardner, "Cyclic Numbers and Their Properties," *Scientific American*, March 1970, pp. 121-124.

2 *DECsystem-10 Assembly Language Handbook*, Digital Equipment Corporation, 1972, pp. 2-56 through 2-57.

3 *VAX-11 Architecture Handbook*, Digital Equipment Corporation, 1982.

4 N. Freed, K. Carosso, and D. Grover, *MATHLIB Project Software Library User's Guide*, Harvey Mudd College, 1982.

5 D. Cortesi, "Dr. Dobb's Clinic – DUP DUP NEG AND =," *Dr. Dobb's Journal*, Vol. 6, No. 9, p. 60.

6 C. Redeker and R. Duncan, *COMPASS – An Assembly Language*, University of Washington, Seattle, Washington, 1977.

7 R. Duncan, "Intel's 8087 Numeric Data Processor," *Dr. Dobb's Journal*, Vol. 7, No. 8, pp. 47-50.

8 M. Gardner, "The Curious Properties of Gray Code and How It Can Be Used to Solve Puzzles," *Scientific American*, August 1972, pp. 106-109.

9 M. Ercegovac, "A Fast Gray to Binary Conversion," *IEEE Proceedings*, April 1978, pp. 524-525.

10 D. Agrawal, "A Comment on Gray to Binary Conversion," *IEEE Proceedings*, March 1979, pp. 444-445.

11 D. Knuth, *The Art of Computer Programming – Combinatorial Algorithms*, Vol. 4 (preprint version), Stanford University, Stanford, California.

12 L. Guibas and J. Stolfi, "A Language for Bitmap Manipulation," *ACM Transactions on Graphics*, Vol. 1, No. 3, pp. 192-214.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 235**

An 8080 Fig-Forth Directory and File System

This article describes a Forth program which overlays a directory and file system on the version of Fig-Forth which has been adapted to the North Star disk system by Interactive Computer Systems, Inc., 6403 Demarco Road, Tampa, Florida 33614. The program requires only one CODE word linkage to the North Star DOS to implement a dozen useful file commands. Except for this one disk-system-peculiar word, the entire file overlay is written in high-level Forth; hence the end product is highly portable.

The article commences with a commands summary in Fig-glossary style. This is followed by a short discussion of the author's rationale for the approach used and a description of the North Star disk and directory formats. The article concludes with a few explanatory notes as an aid in deciphering the appended listing.

Summary of Commands

All commands that prompt for input can be aborted after the prompt occurs by typing a ctrl-c. Input errors which are detected by the operator, as opposed to the command itself, may be corrected by typing a space and re-entering the correct input after the prompting error message. File names and disk names may be from one to eight of any printing ASCII characters from ! to Z inclusive (33 hex to 5A hex inclusive). Note that this differs from the North Star format, which requires alpha and numbers only. The validity of the range of drive numbers (1-3) is checked in all commands where drive number is an input.

The following is a summary of the commands in Fig-glossary style:

CD DR#1 DR#2 . . .

Copies diskette on drive DR#1 to diskette on drive DR#2.

CF DR#1 DR#2 . . .

Copies a named file on drive DR#1 to a named file on DR#2. The names of the files are prompted. The file on drive DR#2 does not have to pre-exist; its directory entry will be created. If it does pre-exist, the file transfer will occur only if the file is sufficiently large. If the file is created, it will be checked for possible disk overflow. If overflow would occur, the command aborts.

CLR-SCR SCR#1 SCR#2 . . .

Clears a block of screens from SCR#1 to SCR#2.

CRF DR#1 . . .

Creates a named file on drive DR#1. Prompts for file name, screen start, screen stop, file type. Will create a directory entry for all file types except a Forth data file (type 5). Forth source program files should be designated as type 4. Type 1 files (object code) will request a hex load-and-go address. In keeping with the general permissiveness of Forth, no check is made for the validity of the screen numbers. Checks for existing file of same name and aborts if found. Existing file entry may be changed by ED command followed by CRF command. Does not check for potential disk overflow and assumption is that all screens of named file are contiguous on the same disk.

CSCR SCR#1 SCR#2 SCR#3 . . .

Copies block of screens SCR#1 to SCR#2 inclusive to block of screens starting at SCR#3.

CSEC SEC#1 SEC#2 DR#1 SEC#3 DR#2 . . .

Copies block of sectors SEC#1 through SEC#2 on drive DR#1 to the block starting at sector SEC#3 on drive DR#2. *Warning*: No check is made to check the block start address SEC#3 for potential overlap with existing named files.

DE DR#1 . . . .

Deletes the specified named directory entry on the specified drive DR#1. File name is prompted. Deletes the directory entry only. Does not access the disk file region in any way; consequently this command may be used to change the file parameters (see CRF command).

FILE #SCRNS DR#1 FILE FILE-NAME

BYTE# DR#1 FILE-NAME

At definition time, FILE creates a named type 5 Forth data file in the directory of the specified drive. When FILE-NAME is executed, the stack will contain the RAM address of the specified byte on the stack. At definition time, a check is made for potential disk overflow. The command aborts on such an eventuality. Likewise, a pre-existing file of that name on the drive will abort the file definition, i.e., duplicate file names on the same drive are not permitted. Insufficient directory space also aborts the command. *Caution*: No check is made on the range of the specified byte. The purpose of this word is to allow disk data files to be created and manipulated in Forth source programs. Files created using this word cannot be accessed by the LIST-FILE and LOAD-FILE commands. (An example using FILE is shown in Figure 1 on this page.)

LI DR#1 . . . .

Lists the directory of the specified drive in North Star format, that is, file name, sector start, file length in sectors, file type, and load-and-go address if file is of type 1.

LI.S DR#1 . . . .

Lists the directory of the specified drive in the format file name, screen start,

```
( create a file 10 screens long on drive #2 )
        10 2 FILE TST-FILE
 ( write ASCII characters into the file )
: WRT 90 33 DO      ( from ASCII ! to Z )
        500 I +        ( form byte address 500+I )
        2              ( drive # )
        TST-FILE     ( invoke file RAM addr )
        I              ( ASCII value to stack )
        SWAP C!       ( store into file )
        LOOP          ( keep going till Z )
        FLUSH         ( be sure file goes to the disk )
;
( read the ASCII characters out to the terminal )
: READ 90 33 DO 500 I + 2 TST-FILE C@ EMIT LOOP ;
```

Figure 1.

by Alfred J. Monroe

Alfred J. Monroe, 3769 Grandview Blvd., Los Angeles, California 90066.

screen stop, file type, and load-and-go address if file is type 1.

LIST-FILE DR#1 . . . .

Used to list the contents of a Forth source program file. Prompts for the name of the file. Checks file type. File type must be 4.

LOAD-FILE DR#1 . . . .

Used to load a named Forth source program file from the specified drive. File type must be 4.

ND DR#1 . . . .

Provides a means of uniquely identifying each diskette as the first entry in the file directory. It reserves the first four sectors of the disk for the directory. This command will overwrite any pre-existing first entry. Prompts for the disk name.

Approach

Several of the commands described above prompt the operator for additional parameters. Some are forgiving in the sense that they provide for error recovery in the event of a syntax error. In the event that the error is of the type that is not caught by the routine, e.g., input of a decimal number that is unintentional, simply enter a space and proceed to enter the correct sequence of digits after the error message. It is for this reason that (word) and (number) are not used in the routines. In addition, the error messages have been made a part of the definition to avoid the necessity of maintaining the master diskette in drive one, as would be the case if "message" had been used.

The North Star DOS provides a number of useful routines through a jump table at the beginning of DOS:

DLOOK	searches directory for a file name.
DWRIT	writes directory back to disk.
DLIST	lists the directory.
DCOM	reads or writes to disk.

In the initial approach to a file system development, maximum use was made of these routines. This approach was ultimately abandoned principally because it severely reduced the portability of the end result. The file system presented here is written entirely in high-level Forth with the exception of the READ/WRITE to the disk. This function is accomplished by means of code word linkage to DCOM. As a result, the current design should prove useful on systems other than the North Star disk system.

Description of Disk and Directory Formats

The North Star disk format consists of 35 tracks of 10 sectors per track for a total of 350 sectors per diskette. They are numbered from 0 to 349 inclusive. Each sector consists of 256 bytes. The first four sectors (= 1024 bytes = 1 screen) are reserved for the directory. There is a maximum of 64 entries. Each entry consists of 16 bytes. Bytes 0-7 are the name made up of from one to eight of any printing ASCII characters and filled with trailing blanks for names less than eight characters long. Note that this generalizes the name convention of North Star. Bytes 8-9 (lo-hi) are the binary decimal value of the disk sector start address of the file. Bytes 10-11 (lo-hi) are the binary decimal value of the length of the file in sectors. Byte 12 is the binary value of the file type. North Star has defined 0 as the default type, 1 for executable object code, 2 for BASIC source programs, and 3 for BASIC data files.

This Forth file system further defines a Forth source program file as type 4, and a Forth data file as type 5. Bytes 13-15 are type dependent. In particular, for a file of type 1, bytes 13-14 (lo-hi) are the hexadecimal RAM address of file-load and start-of-execution, and byte 15 is unused.

The North Star disk and directory formats are adhered to in this Forth file system so that disk directories may be read and utilized in DOS and BASIC. Since Fig-Forth assumes 512 bytes per block, two blocks per screen, manipulation of a directory entry must observe the conversion factors that there are two sectors per block and four sectors per screen.

Deletion of a directory entry is accomplished by replacing the entry with sixteen blanks. The disk file is left untouched; that is, it is recoverable in the event of accidental erasure of its directory entry. Similarly, creation of an entry does not alter the disk elsewhere; hence increasing the length of a file or changing other file descriptive parameters may be accomplished by erasing its name from the directory and recreating the file name with the new file parameters.

Directory entries for Forth source files are created by operator keyboard input of screen start and screen stop. It is assumed that such source files are a set of contiguous screens on the same disk. In keeping with the general permissiveness of Forth, no check is made that the screen numbers are valid. However, all screen numbers are converted to the proper sector numbers.

On the other hand, creation of a Forth data file is inhibited if its length would cause a disk overflow, and the copy file command (CF) will also abort if it would lead to disk overflow.

The fact that the directory is in terms of file sector start and file sector length is not very convenient for use in Forth. Therefore, two ways of listing the directory have been prepared, either in terms of sectors or in terms of screens.

Finally, it should be noted that this file system was generated for a single-density disk system. Accordingly, some changes will be required in word definitions where higher, or mixed, density is involved.

The attached listing (page 42) compiles to 3933 bytes. Somewhat less memory is required if DCOM of screen 89 is not required.

Notes on the Listing

Since the listing is rather sparsely commented, the following is offered as an aid to understanding the file system. In screen 88 a few convenient, but not necessarily essential, constants and variables have been defined. Their main excuse for existence is as a mnemonic aid for rereading the listing six months from now.

Screen 89 defines the only CODE word used in the file system. It links to the North Star DOS routine of the same name in order to perform disk READ/WRITE operations. It is, of course, disk system peculiar. At the time that this system was developed, there was no documented word of similar nature from Omniforth. Since then, Omniforth has circulated a newsletter on RWDSK, a previously undocumented word in Omniforth which behaves in an identical fashion to DCOM. I have left DCOM in the listing in order to illustrate how such a linkage is effected, but am now using RWDSK on my own system. (Why waste memory?)

The input utilities GET-NUM, GET-HEX, and GET-NAME are used to allow error recovery and operator command abort during command prompt input.

The routine ?ENTRY in screen 98 prompts the operator for a file-starting screen, the last screen of the file, and the file type. If the file is type 1, the operator is further prompted for a hexadecimal load-and-go address.

The routine ?EMPTY in screen 99 searches for an empty location in the

directory. It expects to find the RAM address of the directory in the variable SAVE. If an empty location is found, the variable BUFR will contain the RAM start address of the empty location and a true flag will be on the stack. Failure to find an empty location will leave a false flag on the stack.

The routine COMPARE in screen 100 compares two strings for equality. It expects ADDR1 ADDR2 LENGTH . . . on the stack. It leaves a false flag on the stack if the strings are not equal, a true flag if equal; now ADDR1 points to the byte following the string associated with the original value of ADDR1; and ADDR2 has been dropped.

The routine FIND-FILE in screen 101 uses COMPARE to locate the file name in IBUF within the directory. On success there is a true flag on the stack and the variable BUFR contains a pointer to the byte following the located name in the directory. On failure, there is a false flag on the stack. The name was placed in IBUF by GET-NAME.

WRITE-DIREC in screen 101 expects the variable SAVE to contain the pointer to the start of the directory in RAM memory. SAVE was initialized by RD-DIREC as is the drive pointer variable DR. WRITE-DIREC writes the updated directory back to disk. DELETE-ENTRY expects the contents of BUFR to point to the entry to be deleted. The entry is deleted by filling that location with 16 blanks and then writing the updated directory back to disk. DO-ENTRY first checks for an empty location in the directory, and if one is found, moves the contents of IBUF to the location pointed to by the contents of BUFR. The directory is then written back to disk.

NXT-SEC# of screen 104 locates the next free sector address above the "highest" file in the directory. Note that the diskettes are not "squished" when a file is deleted from the directory.

DISK-ALLOT is used by FILE to build a type 5 file entry and put it in the directory.

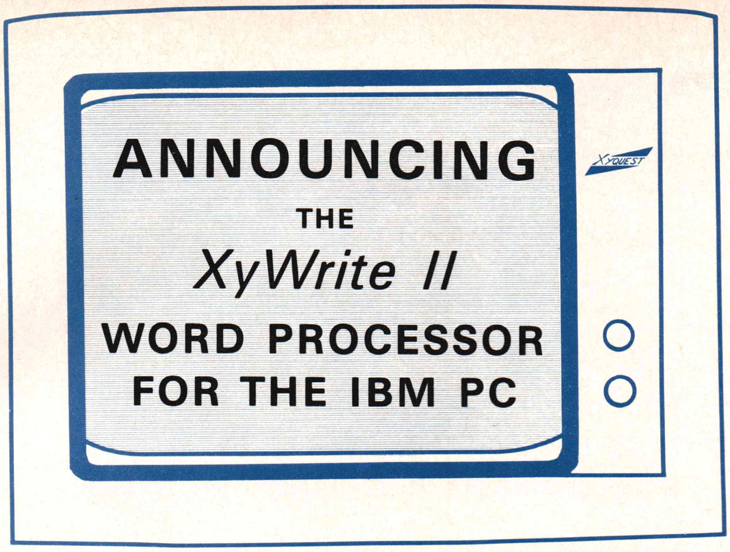
?SPACE of screen 105 is used to make sure that the file being created or written to a disk will not cause a disk overflow.

It is hoped that these brief notes will help the reader to decipher those routines which have not been annotated on the screens.

DDJ

(Listing begins on page 42)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 237**



from XYQUEST

We hereby make you an UNBELIEVABLE OFFER:

For only $7.00 ($5.00 for the disk, and $2.00 for handling), we will send to you a DEMO DISK of XyWrite II, THE ONLY WORD PROCESSOR to take advantage of the SPEED and POWER of the IBM PC.

We believe in our program so we want to make it easy for you to try it. We want you to compare XyWrite II against the rest. You will quickly see the lightning responsiveness of the program: it'll take your breath away.

FASTEST PROGRAM ON THE MARKET

SIMULTANEOUS MULTIPLE FILE ACCESS

SPLIT SCREEN WRITING (HORIZONTAL OR VERTICAL)

SPLIT SCREEN EDITING (MOVE TEXT BACK & FORTH)

HORIZONTAL SCROLLING

EDIT OR WRITE WHILE PRINTING

INTEGRAL MATH PACKAGE

COLUMNAR POSITIONING

INDEXING, FOOTNOTES, HEADERS/FOOTERS

and MUCH, MUCH MORE

Find out what real speed power is on your IBM PC. Compare XyWrite II with WordStar, EasyWriter 1.1 and II, Benchmark 3.0, PowerText, Volkswriter, FinalWord, Select, WordPlus, PerfectWriter - or any of the others.

You have nothing to lose. The high quality diskette is yours to keep. When you decide to buy XyWrite II, you receive a $7.00 discount off the $195.00 list price.
Call or write us today at (617) 275-4439. Ask for John.

WordStar is a registered trademark of MicroPro International. Benchmark is a registered trademark of Metasoft Corp. PowerText is a registered trademark of Beaman Porter, Inc. Volkswriter is a registered trademark of Lifetree Software, Inc. FinalWord is a registered trademark of Mark of the Unicorn. Select is a registered trademark of Select Information Systems. WordPlus, is a registered trademark of Professional Software, Inc. PerfectWriter is a registered trademark of Perfect Software. EasyWriter is a registered trademark of Information Unlimited Software, Inc. IBM and IBM PC are registered trademarks of International Business Machines Corporation.

XyQuest Inc.
P.O. Box 372
Bedford, MA 01730

Circle **no. 18** on reader service card.

Fig-Forth Directory & File System

(Text begins on page 38)

```
SCR # 88
( CONSTANTS, VARIABLES, & CONVERSION FACTORS )
48 CONSTANT "0"   57 CONSTANT "9"   65 CONSTANT "A"
70 CONSTANT "F"   90 CONSTANT "Z"   3 CONSTANT CNTRL-C
HEX 2020 CONSTANT BLBL DECIMAL    ( A DBL BLANK )
0 VARIABLE BUFR   0 VARIABLE FLAG   0 VARIABLE SAVE
0 VARIABLE CNTR   0 VARIABLE DR     0 VARIABLE DW
0 VARIABLE #SEC   0 VARIABLE LAST-SEC   0 VARIABLE SECR
0 VARIABLE SECW
( SCRN2SEC    SCR# ... SEC# )
: SCRN2SEC 87 /MOD DROP 4 * ;
( SEC2SCRN    SEC# DR# ... SCR# )
: SEC2SCRN 1- 87 * SWAP 4 /MOD SWAP DROP + ;    -->
```

```
SCR # 89
( DCOM - LINKAGE TO DISK READ/WRITE ROUTINE OF NORTH STAR DOS )
( DCOM - RAM-ADDR SEC-STRT #SEC R/W DR# ....        )
( R/W=1 FOR READ, =0 FOR WRITE                      )
HEX CODE DCOM
    I' L MOV   I H MOV        ( PRESERVE II' )
    XTHL  L I' MOV            ( DR# TO C )
    H POP XTHL  L I MOV       ( R/W TO B )
    H POP XTHL  L A MOV       ( #SEC TO ACC )
   H POP XTHL  XCHG         ( DSK ADDR TO DE )
   H POP XTHL  XCHG     ( RAM ADDR TO DE, DSK ADDR TO HL )
   2022 CALL  I POP ( CALL DCOM & RSTR II' )
   0 H MVI  CS IF 1 L MVI  ELSE 0 L MVI ENDIF  ( ERROR FLAG )
   H PUSH  NEXT JMP  DECIMAL
-->
```

```
SCR # 90
( LI & LI.S )
( THERE ARE TWO DIRECTORY LISTING COMMANDS, LI & LI.S )
( LI LISTS THE DIRECTORY IN NORTH STAR FORMAT, THAT IS )
( NAME, SECTOR START, FILE LENGTH IN SECTORS, FILE TYPE )
( & GO ADDRESS IF APPLICABLE.  THE LI.S COMMAND REPLACES )
( THE SECTOR INFORMATION WITH SCREEN START & SCREEN STOP )
( DR-TST -       DR# ... DR#  ABORTS IF DR# OUT OF RANGE )
: DR-ERR CR ." DR# ERROR " CR ABORT ;
: DR-TST DUP 1 < IF DR-ERR ELSE DUP 3 > IF DR-ERR
        ENDIF ENDIF ;
: RD-DIREC 0 BUFFER DUP DUP SAVE ! BUFR ! 0 4 1 DR @
     DCOM DISK-ERROR ! ;
-->
```

```
SCR # 91
< LI & LI.S CONTINUED >
: ?SCRN FLAG @ IF DR @ SEC2SCRN ENDIF ;
: LIST-DIREC BUFR @ 16 BUFR +! DUP 8 TYPE 8 + DUP @ DUP
    2 SPACES ?SCRN 3 .R 2 SPACES SWAP 2+ DUP @ ROT
 FLAG @ IF + ?SCRN 1- ELSE DROP ENDIF 3 .R 2 SPACES 2+ DUP C@
    DUP 3 .R 1 = IF 2 SPACES 1+ @ .4H ELSE DROP ENDIF ;
: DOUT BUFR @ @ BLBL = IF 16 BUFR +! ELSE LIST-DIREC CR ENDIF ;
: .LI CR DR-TST DR ! FLUSH EMPTY-BUFFERS RD-DIREC CR 0 CNTR !
    BEGIN CNTR @ 1 CNTR +! 64 < WHILE DOUT REPEAT ;

< LI.S -  DR#... LISTS FNAME,SCR# STRT,SCR# STOP,TYPE,GA >
: LI.S 1 FLAG ! .LI ;

< LI   - DR# ... LISTS FNAME,SEC# STRT,FILE LEN,TYPE,GA >
: LI  0 FLAG ! .LI ;
-->


SCR # 92
< THE CD COMMAND >
< CD    DR#1 DR#2 ...     >
< COPIES ENTIRE DISKETTE ON DR#1 TO DISKETTE ON DR#2 >

: NEXT-SEC #SEC @ DUP SECR +! SECW +! LAST-SEC @ SECR @
           #SEC @ + - DUP 0< IF #SEC +! ELSE DROP ENDIF
           #SEC @ 0= ;


: DWRITE LIMIT SECW @ #SEC @ 0 DW @ DCOM DISK-ERROR ! ;
: DREAD LIMIT SECR @ #SEC @ 1 DR @ DCOM DISK-ERROR ! ;

< ABORT COMMAND ON A CNTRL-C >
: ?ABORT CNTRL-C = IF DROP QUIT ELSE DUP ENDIF ;

-->


SCR # 93
< THE CD COMMAND CONTINUED >
< THE WORD SET-#SEC FINDS THE AVAILABLE NUMBER OF SECTORS >
< ABOVE LIMIT THAT CAN BE USED AS A DISK BUFFER.  NORMALLY >
< THE TOP OF MEMORY IS A CONSTANT, BUT NOT IN MY SYSTEM!! >
: SET-#SEC LIMIT
      BEGIN 256 + DUP DUP 0 SWAP ! @
-1 =  UNTIL
      256 - LIMIT - 256 / DUP 32 > IF DROP 32 ENDIF #SEC ! ;
< THE WORD DCOPY IS COMMON TO CD AND CSEC >
: DCOPY  BEGIN DREAD DWRITE NEXT-SEC UNTIL
       CR ." COPY COMPLETED " CR ;
: CD SWAP DR-TST DUP DR ! ." COPY FRM DSK " . DR-TST
     DUP DW ! ." TO DSK " . CR
     ." HIT SPACE BAR TO CONTINUE, ANY OTHER TO ABORT "
     KEY 32  IF CR 350 LAST-SEC ! 0 DUP SECR ! SECW !
         SET-#SEC DCOPY ELSE ." CMND ABORTED " CR ENDIF ;  -->
```

(Listing continued on next page)

Fig-Forth Directory & File System

(Listing continued, text begins on page 38)

```
SCR # 94
< THE CSEC COMMAND >
< CSEC     SEC#1   SEC#2   DR#1   SEC#3   DR#2   ...       >
< COPIES SEC#1 THRU SEC#2 FROM DRIVE DR#1 TO DR#2   >
< STARTING AT SEC#3. >
: CSEC DR-TST DW ! SECW ! DR-TST DR ! 1+
        LAST-SEC ! SECR ! SET-#SEC LAST-SEC @ SECR @
     -  DUP #SEC @ < IF #SEC ! ELSE DROP ENDIF DCOPY ;
< THE FOLLOWING SEVERAL SCREENS DEVELOP A SET OF UTILITIES. >
< THE WORD BUFF IS USED TO CREATE TEMPORARY HOLDING BUFFERS >
< FOR COMMAND PARAMETER INPUT. >
< BUFF -     N1 BUFF NAME         ALLOTS N1 BYTES TO NAME >
: BUFF <BUILDS ALLOT DOES> ;
16 BUFF IBUF    16 BUFF TBUF    -->

SCR # 95
< GET-NUM  LEAVES DECIMAL # ON TOP OF STACK >
: SYN-ERR CR ." SYNTAX ERROR " DROP DROP 0 "0" CR ;
: GET-NUM  0 FLAG ! < QUIT WHEN FLAG = 1 >
   0 BEGIN  < PUT INITIAL # = 0 ON TOP OF STACK >
          KEY DUP DUP EMIT ?ABORT < GET CHR & ECHO IT >
     13 = IF 1 FLAG ! DROP  < QUIT ON A CR >
        ELSE DUP "0" < IF  < CHR < 0 ? >
                     SYN-ERR
        ELSE DUP "9" > IF  < CHR > 9 ? >
                     SYN-ERR ENDIF ENDIF ENDIF
< NOW CNVT ASCII TO DECIMAL & ADD TO THE STACK >
 FLAG @ 0= IF 48 - SWAP 10 * + ENDIF
  FLAG @ UNTIL ; < KEEP GOING TILL CR >
-->

SCR # 96
< GET-HEX  LEAVE A HEX # ON TOP OF STACK >
: GET-HEX 0 FLAG ! 0 BEGIN KEY DUP DUP EMIT ?ABORT
          13 = IF 1 FLAG ! DROP
             ELSE DUP "0" < IF SYN-ERR
               ELSE DUP "9" > IF DUP "A" < IF SYN-ERR
             ELSE DUP "F" > IF SYN-ERR ENDIF
             ENDIF ENDIF ENDIF ENDIF
 FLAG @ 0= IF 48 - DUP 9 > IF 7 - ENDIF SWAP 16 * + ENDIF
   FLAG @ UNTIL ;
-->
```

(Listing continued on page 46)

Fig-Forth Directory & File System

(Listing continued, text begins on page 38)

```
SCR # 97
   ( GET-NAME )
   : INIT-IBUF IBUF DUP BUFR ! 8 CNTR ! 0 FLAG ! IBUF 16 BLANKS ;
   : NM-ERR CR ." SYNTAX ERROR " CR ." REENTER FNAME   "
           DROP DROP INIT-IBUF ;
   : GET-NAME INIT-IBUF  ." FNAME? "
          BEGIN KEY DUP DUP EMIT ?ABORT
             13 = IF 1 FLAG ! DROP
                ELSE DUP 33 < IF NM-ERR
                              ELSE DUP "Z" > IF NM-ERR
                                             ELSE BUFR @ 1 BUFR
                                              +! C! -1 CNTR +!
                                             ENDIF
                              ENDIF
                ENDIF CNTR @ 0= IF 1 FLAG ! ENDIF
     FLAG @ UNTIL DROP CR ;
   -->

SCR # 98
   ( ?ENTRY AND CLR-SCR )

   : ?ENTRY  ." SCR STRT?  " GET-NUM CR DUP SCRN2SEC
```

```
            IBUF 8 + DUP 2+ BUFR ! !
            ." LAST SCR?  " GET-NUM CR SWAP - 1+ 4 *
            BUFR @ ! 2 BUFR +!
            ." FILE TYPE?  " GET-NUM CR DUP DUP 5 = IF
             ." WRONG FILE TYPE " CR ABORT ELSE BUFR @ C! 1
          BUFR +! ENDIF
     1 = IF ." HEX GO ADDR?  " GET-HEX CR BUFR @ ! ENDIF ;

 ( CLR-SCR  -    SCR#1 SCR#2 .... CLEARS SCR#1 THRU SCR#2 )

 : CLR-SCR 1+ SWAP DO I EDITOR CLEAR FORTH LOOP ;

 -->

SCR # 99
 ( ?EMPTY )
 : ?EMPTY 0 DUP CNTR ! FLAG ! SAVE @ BUFR !
         BEGIN 1 CNTR +! BUFR @ 16 BUFR +!
               @ BLBL = IF -16 BUFR +! 1 FLAG !
                        ELSE CNTR @ 65 = IF
                        1 FLAG ! ENDIF ENDIF
  FLAG @ UNTIL
         CNTR @ 65 = IF 0 ELSE 1 ENDIF ;


 -->

SCR # 100
 ( COMPARE )
 : COMPARE 1 FLAG !
          0 DO DUP C@ ROT DUP C@ ROT
     = IF 1+ SWAP 1+ ELSE 0 FLAG !  ENDIF LOOP
     SWAP DROP FLAG @ DUP 0= IF SWAP DROP ENDIF ;  -->
 ( COMPARE TWO STRINGS ADDR1 ADDR2 CNT ON STACK.
   LEAVE 0 ON STACK IF NOT EQUAL. 1 IF EQUAL & ADDR1 PNTS
  TO CELL FOLLOWING NAME )


 -->

SCR # 101
 ( FIND-FILE WRITE-DIREC DELETE-ENTRY DO-ENTRY )
 : FIND-FILE 0 CNTR ! SAVE @ BUFR ! 0 FLAG !
        BEGIN 1 CNTR +! IBUF BUFR @ 16 BUFR +! 8
              COMPARE
              IF DROP -16 BUFR +! 1 FLAG ! ENDIF
              CNTR @ 65 = IF 1 FLAG ! ENDIF
  FLAG @ UNTIL
  CNTR @ 65 = IF 0 ELSE 1 ENDIF ;

 : WRITE-DIREC SAVE @ 0 4 0 DR @ DCOM DISK-ERROR ! ;

 : DELETE-ENTRY BUFR @ 16 BLANKS WRITE-DIREC ;

 : DO-ENTRY ?EMPTY IF IBUF BUFR @ 16 CMOVE WRITE-DIREC
                   ELSE CR ." DIRECTORY IS FULL " CR QUIT
                   ENDIF ;   -->
```

(Listing continued on page 50)

YOUR IBM® PERSONAL COMPUTER IS ABOUT TO BECOME MORE POWERFUL THAN YOU EVER DREAMED POSSIBLE.

PC'83

SAN FRANCISCO
Friday-Sunday, June 17-19, 1983
Brooks Hall/Civic Center
10:30 AM to 5:30 PM Daily

BOSTON
Saturday-Monday, October 8-10, 1983
(Columbus Day Weekend)
Bayside Exposition Center
10:30 AM to 5:30 PM Daily

Produced by Northeast Expositions Inc., the people who brought you Applefest, CP/M '83 and the National Computer Shows.
826 Boylston Street, Chestnut Hill, Massachusetts 02167, 617-739-2000 or 800-841-7000.
For information on exhibiting at PC '83, please call 800-343-2222.

Circle no. 44 on reader service card.

ANNOUNCING PC '83 THE INTERNATIONAL CONFERENCE AND EXPOSITION FOR PC USERS

In the last few months, *thousands* of exciting new products have become available to make your IBM Personal Computer work in hundreds of innovative, landmark applications. By adding the right software, plug-in cards, peripherals and accessories, you can make your PC more productive and useful than you ever dreamed possible.

And now there's finally a way to learn about and try out all of these products: PC '83, the first international event *exclusively* for PC users.

THE LARGEST EVER

The exposition portion of PC '83 is the largest gathering of PC-compatible products ever held. You'll find, under one roof, more than 400 exhibits of software for business, professional, home, personal and education applications; plug-in cards for memory expansion, telecommunications, speech and many other functions; hundreds of peripherals, including printers, networking systems and innovative new storage devices; plus dozens of publications, books and invaluable support services.

PC '83 brings together the *entire* PC world.

RELIABLE AND COST-EFFECTIVE

Think about the different vehicles available for buying software and add-ons for your PC. You can shop at local computer stores, but they usually have a very limited selection of products and a less-than-expert staff. You can buy through mail order—and take some big risks if you get products that don't work or don't do what you expected.

At PC '83 you have the best of all worlds. You can see *all* the different PC-compatible products in one location at one time. You can talk with factory experts rather than just local salespeople. You can 'test drive' each product and decide if it's right for you. And when you find what you're looking for, you can buy it right on the spot.

Best of all, you'll save hundreds—even thousands—of dollars at PC '83. Most of the products shown at PC '83 are sold at special show prices, so you won't have to look hard to find incredible values. And because PC '83 will help you make more informed purchase decisions, you'll end up with products that are better suited to your applications.

A REMARKABLE CONFERENCE PROGRAM

The conference program at PC '83 isn't just a warmed-over, pared-down version of other computer conferences. The seminars, workshops and forums at PC '83 are the result of months of research and planning targeted toward one specific goal: helping you get absolutely the most out of your IBM Personal Computer.

The program features world-renowned PC authorities speaking at over 100 educational sessions. Seminars provide an in-depth, understandable look at a broad range of PC uses in business, home and education. Seminars emphasize 'how to,' telling you specifically what you need to know—in non-technical language—to use the PC in dozens of different applications.

Forums present an inside look at what's happening in the PC-compatible industry and what new developments you can expect in the next few years.

Software and Hardware Spotlights are a PC '83 exclusive. These workshops address one of the biggest problems that virtually all computer users face: how to decide which software and hardware packages are best for your applications. Each Spotlight provides a detailed discussion and demonstration (with large-screen television) of a group of products, covering their features, capabilities *and* limitations. Experts are on hand at each of these sessions to answer all your questions.

IBM is a registered trademark of International Business Machines Corp.

EASY ON YOUR FEET

We've thought of a lot of details to make your visit to PC '83 just a little more pleasant. As with all of our events, the show is fully carpeted, so your feet won't give out before you've seen all the exhibits.

Our unique Conference and Exhibits Guide helps you quickly find the exhibits and educational sessions you want to attend—even if you can't remember their names. And the Guide is provided free of charge to all attendees.

If you're traveling some distance to attend PC '83, we'll arrange a discounted hotel room near the show site through the PC '83 Housing Bureau. On request, we will also provide information on things to do, places to visit and where to eat in San Francisco and Boston—to make your stay in these grand cities a memorable and relaxing one.

A TOTAL SUPPORT SERVICE

Let's face it, getting support for your PC is tough. The PC magazines, newsletters, user groups and retailers are certainly helpful. But there are so many different products, services and things to learn about for your PC, what you really need is *one* event that brings everything together in one place in a well-organized format.

PC '83 does it all. It showcases all the PC-compatibles under one roof. It has a complete and comprehensive program of seminars, forums and product workshops. And it's produced with the quality and professionalism you've come to expect from Northeast Expositions.

So if you do anything with the IBM Personal Computer or if you're considering buying one, be sure to put PC '83 in your calendar right away.

Your PC will certainly appreciate it.

DAILY REGISTRATION FEES

Exhibits-only badges are $8 per day, and the Conference Program is $15 per day.

SPECIAL PRE-REGISTRATION DISCOUNT

If you plan to attend PC '83 save now with advance registration. Three-day Exhibits and Conference badges are $48, you save $21. Three-day Exhibits only badges are $18, you save $6. One-day Exhibits and Conference badges are $23 and one-day Exhibits only badges are $8.

ADDITIONAL INFORMATION

To receive additional information about attending or exhibiting at PC '83, including the Conference, Seminar, Workshop and Panel Discussions Program, or information on local hotels call 617-739-2000 or 800-841-7000 (Boston).

ANY QUESTIONS? CALL 800-841-7000 (BOSTON)

Produced by Northeast Expositions, nationwide producers of the National Computer Shows, Applefest and CP/M '83, 826 Boylston Street, Chestnut Hill, Massachusetts 02167.

Fig-Forth Directory & File System

(Listing continued, text begins on page 38)

```
SCR # 102
( DE )
( DE -      DR# ...  PROMPTS FOR FNAME )
( USED TO DELETE NAMED FILES FROM THE DIRECTORY OF THE )
( SPECIFIED DRIVE DR# )

: CHK-NAME DR-TST DR ! RD-DIREC GET-NAME CR FIND-FILE ;
: DE CHK-NAME IF DELETE-ENTRY ELSE ." FILE NOT FOUND "
     CR ENDIF ;
-->

SCR # 103
( LIST-FILE )
( LIST-FILE    -      DR# ... PROMPTS FOR FILE NAME  )
( USED TO LIST NAMED FILE FROM THE SPECIFIED DRIVE )
( WILL ACCEPT FORTH SOURCE PROGRAMS OF TYPE 4 ONLY )

( : .FF 12 EMIT ; )
: LIST-FILE CHK-NAME IF BUFR @ DUP 12 + C@ 4 = IF .FF 8 +
               DUP @ 4 / DR @ 1- 87 * + SWAP 2+ @ 4 /
              0 CNTR ! 0 DO CNTR @ 3 = IF .FF
              0 CNTR ! ENDIF DUP LIST 1+ 1 CNTR +! LOOP
          DROP .FF ELSE DROP ." WRONG FILE TYPE " CR
```

```
          QUIT ENDIF ELSE CR ." FILE NOT FOUND " CR ENDIF ;
-->
```

```
SCR # 104
( NXT-SEC# )
: NXT-SEC# RD-DIREC 0 DUP DUP CNTR !
         BEGIN 1 CNTR +! BUFR @ 16 BUFR +!
               DUP @ BLBL = IF DROP
               ELSE 8 + DUP @ SWAP 2+ @ + DUP ROT
               > IF SWAP DROP DUP ELSE DROP DUP ENDIF
               ENDIF CNTR @ 65 = UNTIL DROP ;
: DISK-ALLOT IBUF 16 BLANKS IBUF 8 + !
   4 * IBUF 5 OVER 12 + C! 10 + !
   DUP 1+ SWAP C@ 31 AND
 IBUF SWAP DUP >R CMOVE R> IBUF + 1- DUP C@ 127 AND SWAP C!
    FIND-FILE IF CR ." FILE NAME CONFLICT " CR QUIT ELSE
  DO-ENTRY ENDIF ;
-->
```

```
SCR # 105
( FILE )
( #SCRNS DR#   FILE   FNAME )
( BYTE# DR#   FNAME      LEAVES RAM ADDR OF BYTE# )
( ?SPACE      #SCRNS SEC-STRT ... #SCRNS SEC-STRT 1 )
(              #SCRNS SEC-STRT ... 0              )
: ?SPACE 2DUP SWAP 4 * + 350 > IF DROP DROP DROP 0
                                ELSE 1 ENDIF ;
: FILE DR-TST DR ! HERE SWAP NXT-SEC# ?SPACE
  0= IF ." INSUFFICIENT DISK SPACE " CR QUIT ENDIF
     <BUILDS  DUP , DISK-ALLOT
     DOES> @ SWAP DUP DR ! 1- 174 * SWAP 2 / +
           SWAP 512 /MOD ROT + BLOCK + UPDATE ;
-->
```

```
SCR # 106
( ND )
( ND  -    DR# ... PROMPTS FOR DISK NAME  )
( PROVIDES A MEANS OF UNIQUELY IDENTIFYING EACH DISK )
( AS THE FIRST ENTRY IN THE DIRECTORY.  IT RESERVES )
( THE FIRST 4 SECTORS OF THE DISK FOR THE DIRECTORY. )
( THIS COMMAND WILL OVERWRITE ANY PRE-EXISTING FIRST ENTRY. )
: ND DR-TST DR ! RD-DIREC GET-NAME IBUF SAVE @ 8 CMOVE
     SAVE @ 8 + DUP 0 SWAP ! 2+ DUP 4 SWAP !
     2+ DUP 0 SWAP C! 1+ 0 SWAP ! WRITE-DIREC ;
-->
```

(Listing continued on next page)

Fig-Forth Directory & File System

(Listing continued, text begins on page 38)

```
SCR # 107
  ( CSCR )
  ( CSCR   -   SCR#1 SCR#2 SCR#3 ...                    )
  ( COPIES FROM SCR#1 THRU SCR#2 TO SCR#3 & UP          )
  : CSCR SECW ! 1+ SWAP DO I SECW @ 1
       SECW +! EDITOR COPY FORTH LOOP ;
  -->

SCR # 108
  ( CRF )
  ( CRF   -     DR# ...   PROMPTS FOR OTHER PARAMETERS    )
  ( USED TO CREATE DIRECTORY ENTRIES FOR ANY FILE TYPE )
  ( EXCEPT TYPE 5, THAT FORTH DATA FILES ARE INHIBITED. )
  : CRF DR-TST DR ! RD-DIREC GET-NAME ?ENTRY FIND-FILE
         IF CR ." FILE ALREADY EXISTS " CR
```

```
        ELSE DO-ENTRY ENDIF ;
-->

SCR # 109
( LOAD-FILE )
( LOAD-FILE   -    DR# .... PROMPTS FOR FNAME        )
( USED TO LOAD FORTH SOURCE PROGRAM FILES.           )
( ACCEPTS TYPE 4 FILES ONLY. )
: LOAD-FILE CHK-NAME
          IF BUFR @ 8 + @ DR @ SEC2SCRN LOAD
            ELSE CR ." FILE NOT FOUND " CR ENDIF ;
-->

SCR # 110
( CF  )
( CF  -    DR#1 DR#2 ... PROMPTS FOR FILE NAMES    )
( COPIES SPECIFIED FILE FROM DRIVE DR#1 TO SPECIFIED )
( FILE ON DRIVE DR#2.  FILE NEED NOT EXIST ON DR#2,  )
( SINCE ROUTINE WILL CREATE THE NECESSARY ENTRY IF )
( THERE IS ROOM ON THE DISK.  IF THE FILE ALREADY  )
( EXISTS ON DR#2 THEN THE COPY PROCEEDS ONLY IF THE )
( FILE IS SUFFICIENTLY LARGE. THAT IS >= )
```

(Listing continued on next page)

Fig-Forth Directory & File System (Listing continued, text begins on page 38)

```
 0 VARIABLE DT    0 VARIABLE DF          16 BUFF FBUF
 : WRITE-SEC FBUF 8 + DUP @ ( START SEC )
         DUP ROT 2+ @ + 1-         ( LAST SEC )
         DF @ ( FROM DRIVE )
         TBUF 8 + @ ( FIRST TO-SECTOR )
         DT @ ( TO-DRIVE )
         CSEC ( XFR SECTORS ) ; -->

SCR # 111
 ( CF CONTINUED )
 : CF SWAP DUP ." FROM DRIVE " . ." & FROM " CHK-NAME
 IF DR @ DF ! BUFR @  FBUF 16 CMOVE ( ENTRY TO FBUF, DF )
    ELSE ." FILE NOT FOUND " CR QUIT ENDIF
  DUP ." TO DRIVE " . ."  & TO  " CHK-NAME
  IF DR @ DT ! BUFR @  TBUF 16 CMOVE ( ENTRY TO TBUF,DT )
        TBUF 10 + @ FBUF 10 + @ 1- - ( LEN DIFF => 0 IS OK )
        IF WRITE-SEC QUIT ENDIF
    ELSE ( CREATE A NEW ENTRY ON TO-DRIVE )
        DT @ DW ! FBUF IBUF 16 CMOVE NXT-SEC# DUP IBUF 8 + !
       IBUF 10 + @ + DUP 350 > IF ." INSUFFICIENT DISK ROOM "
                                CR QUIT
                             ELSE DO-ENTRY ENDIF
         ( NOW WRITE THE SECTORS TO THE DRIVE )
       IBUF TBUF 16 CMOVE WRITE-SEC
  ENDIF ;
```

End Listing

SAY"
Forth Votrax Driver"

Forth enthusiasts frequently cite modularity and expandability as the language's greatest strengths. The truth of these claims can be shown with an example: Design of all necessary software to interface a Votrax Type'N Talk speech synthesizer to a system.

BASIC and other "conventional" languages make such software interfaces difficult to design and inconvenient to use. In general, all or most of the interface routine must be written in machine language; portability between systems, even within the same processor family, is rare. The routine must be explicitly included in every program requiring use of the peripheral, by either keying in a subroutine or loading a utility module from mass storage. Commands within the BASIC program (let alone the machine code routines) are generally not self-documenting, with obscure references like K=USR(7) or X=INP(231) AND &127 perhaps meaning to initialize the proper port or check an input bit.

With Forth, though, the entire software interface can be programmed in portable high-level code, with virtually no penalty to execution speed and memory requirements. The resulting interface is hardly an inelegant "patch job," as in BASIC; after development, the interface becomes part of the language, usable in exactly the same manner as any other Forth "words." It is even possible to make the interface software part of the core language, so that it will be available immediately upon a cold start without an additional loading operation. Perhaps best of all, interactive development and testing make the software design process incredibly fast and efficient – in this instance, the Votrax software was developed from first idea to final test in under 20 minutes, and still uses less than 100H bytes of memory.

The basic function of such an interface routine is to move data from memory to the peripheral, in accordance with the data conventions of both the language and the device. The Votrax Type'N Talk is designed to receive ASCII data via an RS-232C port, using any of eight baud rates. At low data speeds, and with short and relatively infrequent output strings, the physical interface does not need handshaking lines. Longer, faster, or more frequent output, though, will overflow the Votrax's limited buffer unless handshaking is implemented using the Type'N Talk's Clear To Send line. In this instance, Clear To Send is simply brought into a single control port bit on a Vector Graphics serial I/O board; other systems may use different handshaking schemes appropriate to the hardware.

The interface routine in Listing 1 uses the Forth Interest Group's Forth (Fig-Forth) for the 8080, release 1.1. With minor changes at most, the software should be portable to other Fig-Forth installations regardless of processor types. Only CLEARTHROAT and two or three integer constants in SAYCHARACTER are system-dependent. In this case, CLEARTHROAT is defined to output the six data bytes shown in line 4 of screen 500 to serial control port 07H. These bytes initialize serial ports 06H and 07H of the I/O board for the appropriate word size and baud rate. If the system uses serial ports which are integral to the system or initialized upon cold start, CLEARTHROAT or its equivalent may not be necessary. In any event, CLEARTHROAT is only executed once, during the loading operation.

The core Fig-Forth language, available after a cold start, contains over a dozen words for terminal output. Like most of the language, the "building block" approach has been used extensively in these word definitions. In most instances, only EMIT (which takes an ASCII character byte from the stack and sends it to the terminal) is defined as a machine-language "primitive;" all other terminal output builds upon EMIT to output strings and numerics.

The same scheme can be used for output to the Votrax. As defined in lines 11-14 of screen 500, SAYCHARACTER treats the value currently on top of the stack as an ASCII character byte, and outputs it to the Votrax (and optionally to the terminal as well) with full handshaking. No machine code is required, as the core Fig-Forth language already provides all necessary operations in high level.

SAYCHARACTER executes by first entering an indefinite BEGIN. . .UNTIL loop to read input from port 07H, repeating until a Clear To Send signal is received. For the hardware used here, bit 0 on port 07H is the handshaking signal; a zero on that bit signals that the Votrax's input buffer is full. The loop will terminate whenever bit 0 goes to a 1. Line 13 duplicates the top stack value for later use, then outputs the copy to the Votrax via serial port 06H.

For convenience, one additional feature is added to the definition of SAYCHARACTER. At times, it may be necessary to send output to both the Votrax and the terminal. The most efficient means is to include a software switch in the lowest-level output definition; in that way, all later words defined with this "building block" will use the same method to direct output, at no additional software cost. Here, SAYCHARACTER fetches the value of the variable VOICE+PRINT for use as a logical flag. A non-zero value (a true flag) sends the remaining copy of the character byte to the terminal, using EMIT for output. If the flag value is zero (false), the remaining copy is simply dropped from the stack.

Screens 501 and 502 contain the remaining high-level words to incorporate ASCII string audio output capabilities to the Forth system; all are system-independent, as they are defined using SAYCHARACTER. For ease of use, the words are defined similarly to the Fig-Forth core words for terminal output, with the same stack values and formats.

SPEAK, like TYPE for terminal output of strings, requires two operands from the top of the stack. The top value *n* is the character count of the string whose first data byte is at address *addr*. Any non-zero count will fetch *n* ASCII characters one-by-one from memory to the stack, and then send each to the Votrax. If the count byte is zero, SPEAK simply removes the address operand without output of any characters. Since output of individual characters uses SAYCHARACTER, each may also be directed to the terminal by first storing a true flag value in the VOICE+PRINT variable. Note that any character string in memory can be output using SPEAK, so long as the initial data byte address and character count are first placed on the stack.

SAY" and (SAY") operate in tandem during various phases of operation, in the same manner as the terminal output word ." and its (.") run-time procedure. Character strings delimited by a trailing "quote" character (22H = 34 decimal) are input from the terminal, separated from SAY" by one or more blanks. Further execution of SAY" depends upon Forth's

by C. Kevin McCabe

C. Kevin McCabe, 115 South LaSalle, Suite 1200, Chicago, IL 60603.

current operating state; a zero value for the user variable STATE indicates that Forth is executing rather than compiling.

In the execution state, lines 10-12 of screen 502 move the input string from the terminal buffer to the top of the dictionary, with a leading one-byte character count. The current dictionary pointer is placed on the stack, then used by COUNT to increment the pointer and fetch the contents of the count byte. These two values are the operands for SPEAK, which output the string to the Votrax. Again, VOICE+PRINT can be used to direct output to the terminal as well.

When SAY" is encountered as a component word during compilation of colon definition – that is, in Forth's compilation state – the action is somewhat more complex. SAY" is defined as an immediate word, which executes even during the compilation state. It does not, though, cause immediate output as during execution. Instead SAY" first compiles an address pointer to the (SAY") run-time routine. The text string, along with its leading count byte, is moved to the next available dictionary locations. Finally, the dictionary pointer is incremented to the byte beyond the end of the string.

During later execution of the colon-defined word, execution of (SAY") will output the compiled string. Fig-Forth uses the return stack to hold an address pointer to the next component of a colon-defined word; in this instance, the next "word" is actually the string compiled into the definition by SAY". Lines 11 and 12 of screen 501 non-destructively copy the pointer to the string from the return stack and increment it to designate the first data byte. That pointer is left on the stack, along with two copies of the character-count byte. One copy of the count is incremented (to account for the count byte itself), and used in turn to increment the return stack's word pointer to beyond the string. Finally, SPEAK is used to output the string.

That's it – all the basic building blocks needed for audio output of ASCII strings with the Votrax Type'N Talk. Listing 2 defines several other words using these fundamental components, for convenient output of numeric values and ASCII text from disk screens. Again, all are completely portable, and all use VOICE+PRINT as a software switch – the direct result of Forth's building block approach.

SAYLINE corresponds to Fig-Forth's .LINE and outputs line *n1* of screen *n2* to the Votrax. Similarly, SAYSCREEN outputs an entire screen – presumably ASCII text, since Forth source code's abbreviations and symbols don't translate well to audio.

Single- and double-precision numeric stack values may be output to the audio system with SAYU, SAYN, and SAYD, respectively, in the same manner as U., ., or D. for printed output. If numeric output to the terminal is desired as well, it is easiest to use separate voice and terminal output words in this instance rather than the VOICE+PRINT software switch. As defined, SAYD outputs a negative sign as the word "negative" rather than a single character. In addition, digit characters are output separated by blanks; otherwise, the Votrax would attempt to pronounce a value such as 0CABH as a single word rather than the desired "see-a-bee."

Aside from any necessary port initialization, only SAYCHARACTER will require modification for use on other systems. Appropriate values should be used for the control port number in line 12 and data port number in line 13 of screen 500. If the Votrax's Clear To Send control signal is not brought onto the I/O board as bit 0, it will also be necessary to change the masking constant preceding the logical AND operation in line 12.

Users of the proposed Forth-79 version of the language should note that more extensive changes will be necessary. Some Forth-79 words are defined somewhat differently than their Fig-Forth counterparts; in other instances, no Forth-79 counterpart exists. Examples of the latter problem are the words P! and P@, which transfer data between the stack and I/O ports. Fortunately, most Forth-79 implementations are also supplemented with these and other useful Fig-Forth words.

The Forth-79 version of WORD moves text strings in much the same manner as the Fig-Forth version, but also leaves a pointer to the leading count byte on the stack; remove HERE from both locations in the definition of SAY" and use the address left by WORD instead. Also, the initial value of VOICE+PRINT should not precede the definition, as Forth-79 variables must be initialized after being defined. Finally, Forth-79 does not explicitly require the same type of return stack usage as Fig-Forth (although as a practical matter, few Forth-79 systems will differ). Some experimentation may be necessary to determine if (SAY") will operate in the proper manner.

DDJ

(Listing begins on page 58)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 239**

SAY" (Text begins on page 55)

Listing One

```
Screen # 500
( Type'N Talk Output-1                          CKM 06 Oct 82      )
HEX
: CLEARTHROAT ( initialize Vector Graphics I/O port)
    27 0CE 40 0 0 0 ( data bytes)
    6 0 DO 7 P! LOOP ( output 6 bytes to port 7) ; DECIMAL
CLEARTHROAT ( execute to initialize port)
0 VARIABLE VOICE+PRINT ( 0=voice only, 1=say and print too)
: SAYCHARACTER ( c --> )
    BEGIN 7 P@ 1 AND UNTIL ( wait for clear to send)
    DUP 6 P! ( say the character)
    VOICE+PRINT @ IF EMIT ELSE DROP THEN ( print if desired) ;
-->
Screen # 501
( Type'N Talk Output-2                          CKM 06 Oct 82      )
: SPEAK ( addr n >> )  ( based on TYPE)
    -DUP ( duplicate non-zero count byte)
    IF ( valid count) OVER + SWAP ( find loop parameters)
        DO I C@ ( get next character)
        SAYCHARACTER ( output character) LOOP
    ELSE ( zero count) DROP
    THEN ;
: (SAY")  ( run-time routine based on (.")
    R ( copy pointer to next word) COUNT DUP ( get byte count)
    1+ R> + >R ( increment pointer to next word)
    SPEAK ( output string) ;
-->
Screen # 502
( Type'N Talk Output-3                          CKM 06 Oct 82      )
DECIMAL
: SAY" ( immediate word based on ." )
    34 ( Ascii of " delimiter)
    STATE @ ( fetch compilation state flag)
    IF ( compiling) COMPILE (SAY") ( compile run-time pointer)
        WORD ( move string to top of dictionary)
        HERE C@ ( fetch count byte) 1+ ALLOT ( increment dp)
    ELSE ( executing) WORD ( move string to top of dictionary)
        HERE COUNT ( leave address & count)
        SPEAK ( output string)
    THEN ;
IMMEDIATE
;S
```

Listing Two

```
Screen # 503
( Type'N Talk screen utilities                  CKM 10 Oct 82      )
DECIMAL
: SAYLINE ( n1 n2 >> )  ( based on .LINE)
    (LINE) ( move screen n2 to buffer, leave starting addr and)
           ( character count of line)
    -TRAILING ( adjust count for trailing blanks)
    SPEAK ( output the line) ;
```

```
: SAYSCREEN ( n >> ) ( output Ascii text of entire screen n)
    SCR ! ( save screen number in user variable)
    16 0 DO ( set up line count loop)
       I SCR @ ( leave line, screen numbers)
       SAYLINE ( say the line)
    LOOP ;
;S
```

```
Screen # 504
( Type'N Talk numeric utilities                   CKM 10 Oct 82      )
DECIMAL
: SAYD ( d >> ) ( based on D.)
    DUP ( copy high bytes)
    0< IF ( d < 0 ) SAY" NEGATIVE" THEN ( output sign)
    DABS <# #S #> ( convert to Ascii, leave start addr and n)
    OVER + 1+ SWAP ( get limits for char. by char. output loop)
    DO 32 SAYCHARACTER I C@ SAYCHARACTER 32 SAYCHARACTER
    LOOP ( output with blanks between characters) ;

: SAYN ( n >> ) ( based on . )
    S->D ( convert n to equivalent d) SAYD ( output d) ;

: SAYU ( u >> ) ( based on U.)
    0 ( leave a dummy high-order half of d) SAYD ( output d) ;
;S
```

End Listings

A TRS-80 8080 to Z80 Translator

Back in the May 1981 issue of *DDJ*, No. 45, Robert W. Dea offered us his program "An 8080 to Z80 Translator System." Following is a description of the program and the many modifications I have made to it that will allow the creation of an 8080 source file, and through translation, will produce a Z80 source file that will be accepted by Radio Shack's, or any equivalent, editor/assembler.

A rather large project I had been considering for some time was to convert the Forth Interest Group's 8080 version of Forth into Z80 code. Since I had never bothered to learn the 8080 mnemonics, and was well versed in the Z80 codes, I thought a translated version would afterward allow easier debugging and modification of the Forth program. Since my system is a TRS-80, 48K, model I, with a single disk, I had to modify Mr. Dea's translation program to work with Microsoft's BASIC.

The program was also modified extensively to speed up the translation process. By changing his READ statements into a Data Array, and by using a binary search, as well as using a double-speed modification on my system, and also compressing the BASIC program to get rid of spaces and REM statements, I was able to substantially reduce the time it took to convert the 8080 code to Z80 code. For instance, using a test file of 8080 code from POP H to XTHL, I was able to change the translation time from 7 minutes 39 seconds to 3 minutes 18 seconds.

There are probably many other modifications that could be done to increase the speed even further, such as changing multiple lines to single lines, watching how all the variables are dimensioned, streamlining loops, etc., but the time to do this didn't justify the few seconds more I would get. So I left it the way it is now, and it works fine.

Many parts of the program are just as Mr. Dea wrote it, but I added comments in some lines to help me understand what's going on a little better, and I also wrote a flow chart for it. I found my first round with the program tough going because of the jumps around the BASIC code, but now my understanding is a little better, and I have included the flow chart (starting on page 63) so that you can more easily modify the program if you need to.

Let me go through the program somewhat to show you what I did, and to indicate how an editor/assembler file is created. I should start by describing the part of the program that will create the 8080 code file, so jump to line 4810.

Anthony T. Scarpelli

Anthony T. Scarpelli, 98 Foxcroft Drive, Scarborough, ME 04074.

This program is based on Robert W. Dea's original 8080 to Z80 translator, published in DDJ No. 45. The parts of his original work that appear here are used with his permission. Mr. Dea has now placed his original translator in the public domain. Mr. Scarpelli's work here is also in the public domain.

Creating the 8080 File

I use Ultrados for my operating system, so the first thing we have to do is make sure a buffer is available; that's the CMD"O" in line 4830. With TRSDOS you don't have need of this code, and it can be deleted. I don't know what has to be done with other DOSs.

The instructions are pretty self-explanatory; the only important thing I should note is that this routine does no editing. The only chance to correct mistakes is when you are asked "SURE?". You are given a chance to change your mind by inputting an "N." When you are rolling along, and enter a lot of code, sometimes you forget to correct mistakes. Generally this causes no harm; the translation will just ignore an opcode that can't be found, or something weird will be added to the translation. This can be normally edited out when the file is in the editor/assembler.

The other important point I must mention is due to the way random access files are written to disk. Only 255 characters are allowed in one record, but there is room for 256. The 256th character is a random character of some type in the Z80 EDTASM source file that can be edited out during the assembly process. It could also be edited out from disk with a zapper program of some type, but usually this is not necessary. When using the EDTASM in the assembly mode, the assembly process can be switched to stop on any errors. This is where all editing of the file can be done. Some day I'll see if this can be fixed. As you can see there is still room for some improvements to this program.

Line 5020 adds an up-arrow (CHR$ (91)) delimiter to the file's end marker. All entries have this delimiter. Lines 5030 and 5040 open the random access file, and set up A$ as the buffer.

When we are adding 8080 code to the file, the only difference between a line label or comment line, and an operation code is whether there is a space in the first column or not. Line 5070 prints

```
--> ; MEMORY TEST
SURE?
    \
-->  MVI C,Ø1 ;LOAD REPEAT COUNTER
SURE?
    \
--> START LXI SP,ØFFFFH ;LOCATE STACK COUNTER
SURE?
    \
--> LXI
SURE? N
    \
-->  LXI H,ØØØØH ;LOAD START ADDRESS
SURE?
    \
-->
```

Figure 1.
Output from Create routine.

```
ENTER 8Ø8Ø FILE NAME ? MEMTST8Ø/TMP
ENTER Z8Ø DISK FILESPEC ? MEMTSTZ/SOR
ENTER EDTASM SOURCE FILE NAME (6 CHAR. MAX) ? MEMTST
START LINE NUMBERS AT ? 1ØØ

- ØØ1ØØ  - ; MEMORY TEST
ØØ1ØØ ; MEMORY TEST
- ØØ11Ø  -  MVI C,Ø1 ;LOAD REPEAT COUNTER
ØØ11Ø          LD       C,Ø1              ;LOAD REPEAT COUNTER
- ØØ12Ø  - START LXI SP,ØFFFFH ;LOCATE STACK
ØØ12Ø START    LD       SP,ØFFFFH         ;LOCATE STACK
- ØØ13Ø  -  LXI H,ØØØØH ;LOAD START ADDRESS
ØØ13Ø          LD       HL,ØØØØH          ;LOAD START ADDRESS
- ØØ14Ø  -  LXI D,1FFFH ;LOAD TOTAL BYTES
ØØ14Ø          LD       DE,1FFFH          ;LOAD TOTAL BYTES
- ØØ15Ø  -
```

Figure 2.
Output from Translate program.

a down-arrow to mark this column, and line 5080 is where we prompt for an input. See Figure 1 for an example of this process. Note that a down-arrow prints out as a backslash on my printer.

If you were to exceed the 255 character input to a record, you would jump out of the program and get an error statement. To avoid this, lines 5090 and 5100 make sure that you are aware of this and give you a chance to reduce your line somewhat to fit. The record will be written to disk as soon as it exceeds 215 characters and is less than 255. This wastes a few bytes, but is simple to implement.

The rest of the routine takes the input A1$, adds an up-arrow, and adds it to A2$ if the length is right. In line 5200 spaces are padded to A2$ and it is placed into buffer A$. Then it is put onto disk in line 5210. The record number, R, is incremented and we can now jump back to add more 8080 code to the file. An "@EXIT" closes the file.

Now we have a file filled with 8080 code and comments separated by an up-arrow. We take this file and filter it through the translator in the next section.

Translating

Figure 2 (page 60) shows output from the "translator" part of the program. In essence, what the program does is to first get the 8080 instruction, search through an array until it is found in the data, and then create a string of the Z80 code. Next, the 8080 operand is converted to equivalent Z80 code as specified by the subfields in the rest of the data and added to the string. A data line is specified in lines 170-250.

After the menu, we clear string space and dimension the array in line 430. The only reason for line 440 is to make sure the speed goes back to normal if an error occurs. We have to have normal speed during disk accesses. This error trap can be expanded if necessary.

Next we make sure two buffers are available in 460, then comes the array (83 elements long), then some more initialization until line 1410. Here we ask for the same filespec used in the creation of the 8080 source code file. Then we get the filespec used for the Z80 file, and then the source file name, which is required to be only six characters long.

The format of an editor/assembler file is as follows: the first character is a D3, followed by six ASCII characters for the file name. If the name is less than six bytes, the rest is filled up with spaces to make six. Next is the line number, five bytes long, with each decimal number increased by 176 decimal; for example, a 5 becomes a B5H token. Then comes a space, and then the text with a terminator of an 0D. The next line number follows immediately, and so on. The final byte of the file is a 1A.

Lines 1540 to 1590 fill in the name with spaces. The next routine, lines 1620 to 1760, produces two numbers in the form of strings; one is C2$, which is used for printing, and the other, C3$, is for the Z80 output file.

The next routine, lines 1790 to 1880, goes to the disk and gets the next record, line 1810, and then pulls out the 8080 line of code. Remember it is delimited by an up-arrow, CHR$(91). It leaves this routine with the code in I$, which is printed on the screen.

The next few lines, 1910 to 1940, check to see if we are at the end of the 8080 codes. Also, in order to be able to merge the source code in two or more editor/assembler files, it's a good idea to note the last line number so that you can start the next 8080 file with a larger line number.

We have two output buffers, one for printing, 02$, and one that will go to disk, 0$. The routine in lines 1980 to 2000 first checks to see if the first column is a semi-colon. If so, it will just add the text to the buffer. If not, then we go to the subroutine at 4030. Since this is an important routine, let's go to it.

The 8080 code line is delimited by blanks. If the first column is a blank then we know that the line contains no label, so we test for this in line 4040. If there is

a character we read through I$ until a blank is encountered. The label, up to the next blank, is edited into the output buffers from 4060 to 4150. Then we add eight spaces so the printout of 02$ looks like it would in an editor/assembler, and then add a tab to the file buffer to delimit it. If there was no label, then we merely skip the editing part of the routine. Next we jump to a common routine that scans the code line (I$) to get the next code out of it. Let's jump to it for a minute. GOSUB 3470.

The lines from 3520 to 3590 just jump over any blanks in the code line. Then lines 3640 to 3760 accumulate the actual code and leave it in N$. Now return to 4240, and return to 2040. Then GOSUB 4270. A lot of this code is original and does a lot of jumping around as you can see. It's not my style, but I didn't want to spend a lot of time turning it into some structured code.

Lines 4270 to 4350 are the binary search routine I added. It's a typical binary search so I won't go into it. The routine exits pointing to the data from the array ready to be worked on. That is, F is the index into the array DA$. Now we can return to 2110, if there was indeed a match.

The first thing we do is to make sure our output buffers are empty. Then we go back to our subroutine at 4030 that merely looks at the beginning of the next part of the 8080 code and sees a blank. Remember that this blank separates the opcode from its operand. We then skip to 4170 where we add spaces for pretty printing, and a tab for the Z80 code. Then we get the next subfield, which is the operand, and put it in N$. Back to 2150.

Here we jump immediately to 4380, which is a subroutine that puts all of our array data into another small array S1$(x). It leaves with X$ holding the Z80 code and the rest of the array filled with the subfields which do the translation of the operands. See lines 90 to 250 for a review of how these arrays and subfields are set up.

Now back to 2160 where we put our Z80 code into the buffers and add spaces and a tab.

The next job is to read the subfield codes so that the rest of the 8080 code can be properly added to the buffers. Our index L should contain the number of subfields there are. We start a loop that reads through the subfields and goes to the subroutine that produces a match. These subroutines are relatively simple.

The Subfield Subroutines

The <A> subroutine is the longest and most of the others jump into it, so we'll start our journey there at line 2800. The first thing we do is to get our next subfield from the 8080 code. Then we initialize H$ to the appropriate tree string and start to search the string for a match. The search routine at 3820 is simple enough, so I won't dwell on its finer points.

When we find a match we get the next subfield definition in the tree string, line 2920 to 2990, and edit it into the buffers, line 3000. We set our match flag if we hopefully did find it.

If you look at the flowcharts for the rest of the codes, you will see that <G>, <B>, <C>, <D>, and <E> all jump into the <A> subroutine, and that <F> and <S> do their own little thing. They are simple enough not to explain them in detail.

After the return from the subroutines, we end up back at 2370 and then 2410 on a good match. If there are more subfields to translate, we loop back to do the rest. We finally come to 2430 where we null out the small array, and then edit anything else from our 8080 code, such as comments, into our buffers. We end up at 2560 where we jump to 2660.

Get It Out

The routine here adds in our line number, prints it on the screen at 2670, adds a carriage return, and checks to see if our buffer plus the output buffer is less than our file buffer. If the buffer would overflow, it's time to put it onto disk, so we jump to the routine at 4710 that does just that. If not, we add the code to the output buffer, and jump back to 1650 where we increment our line number and start the whole ball rolling again.

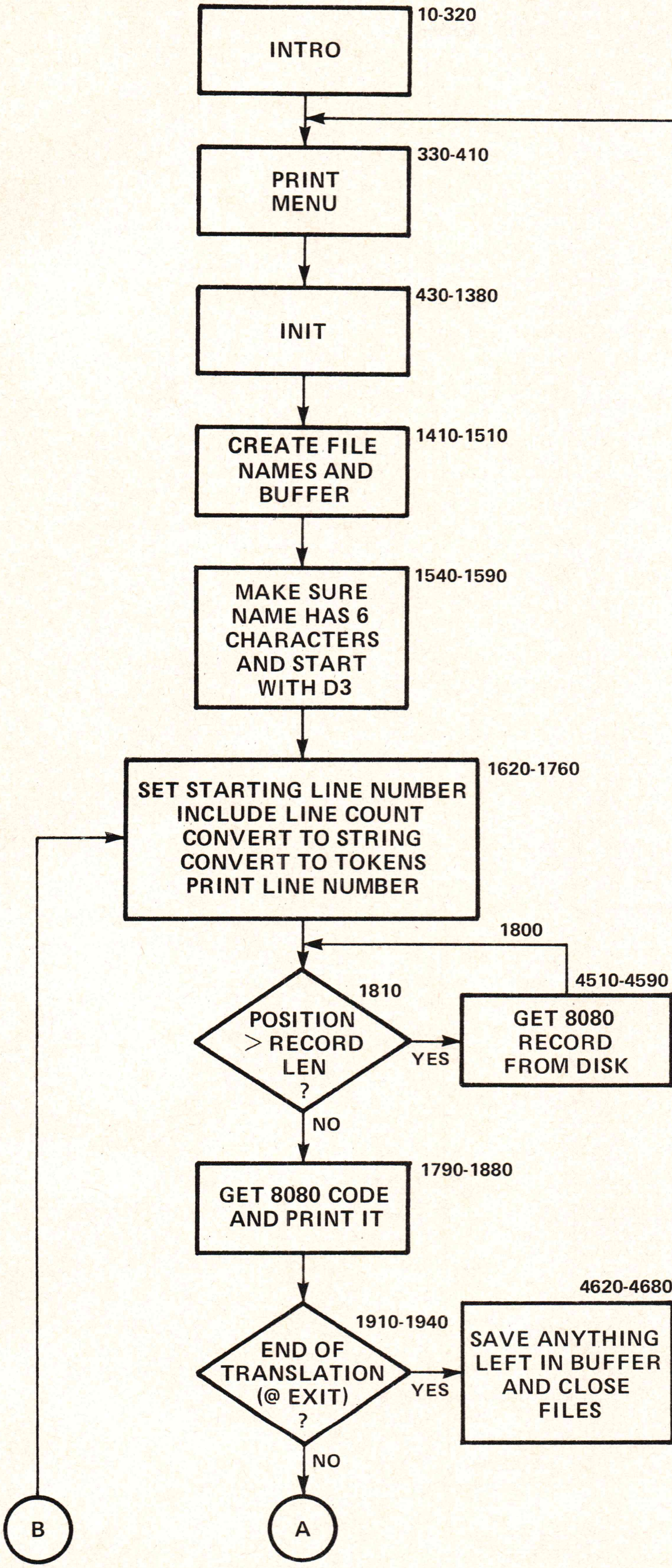
As you can see, all the routines are relatively simple; however, because of all the jumping around, it is difficult to understand and follow without a flowchart. This is a good reason to use more structured programming methods.

The program works, though, and if you need to get those old 8080 files into Z80 code, this is the program to use.

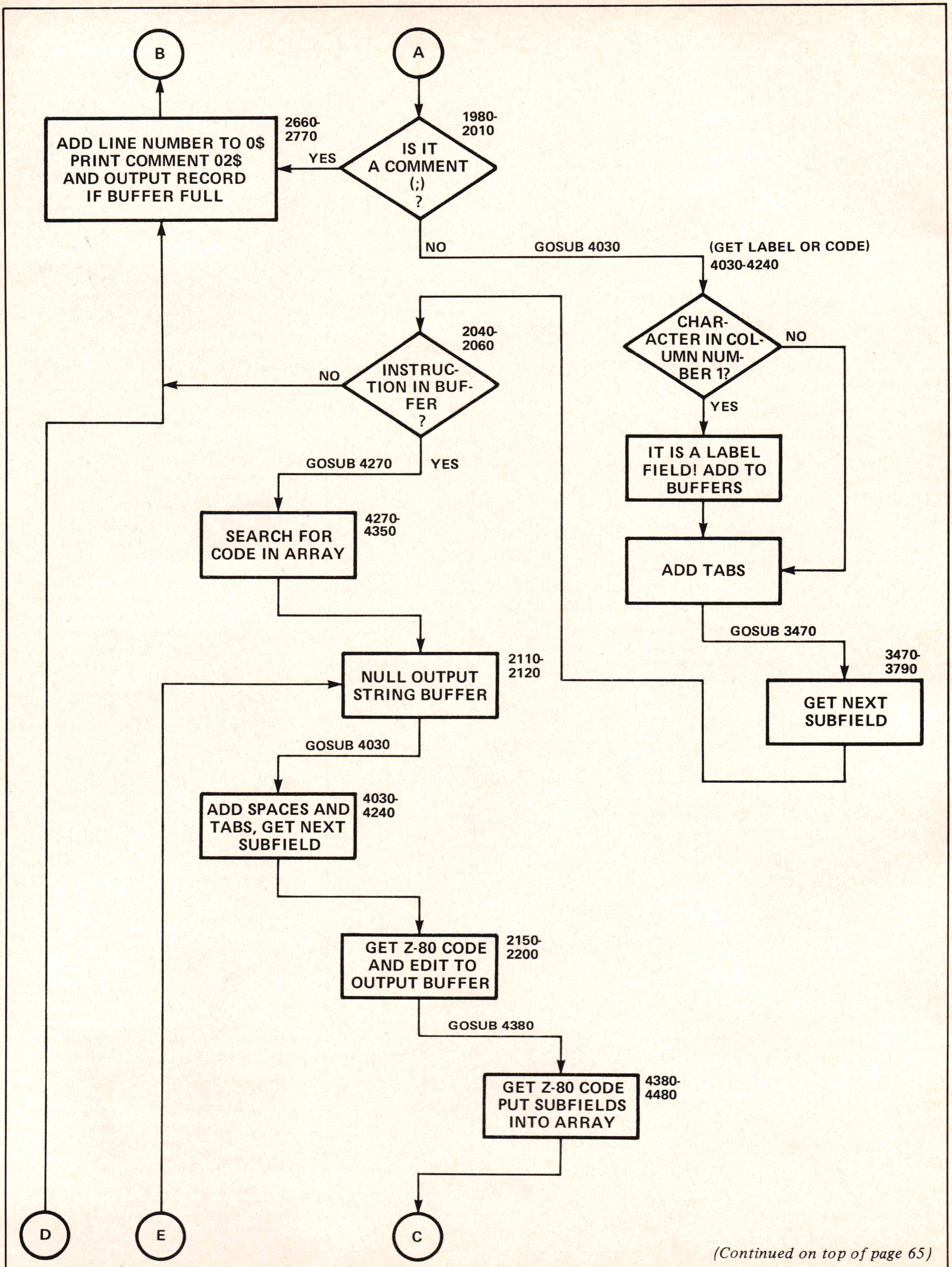
(Flow charts from pages 63-67)

(Listing begins on page 68)

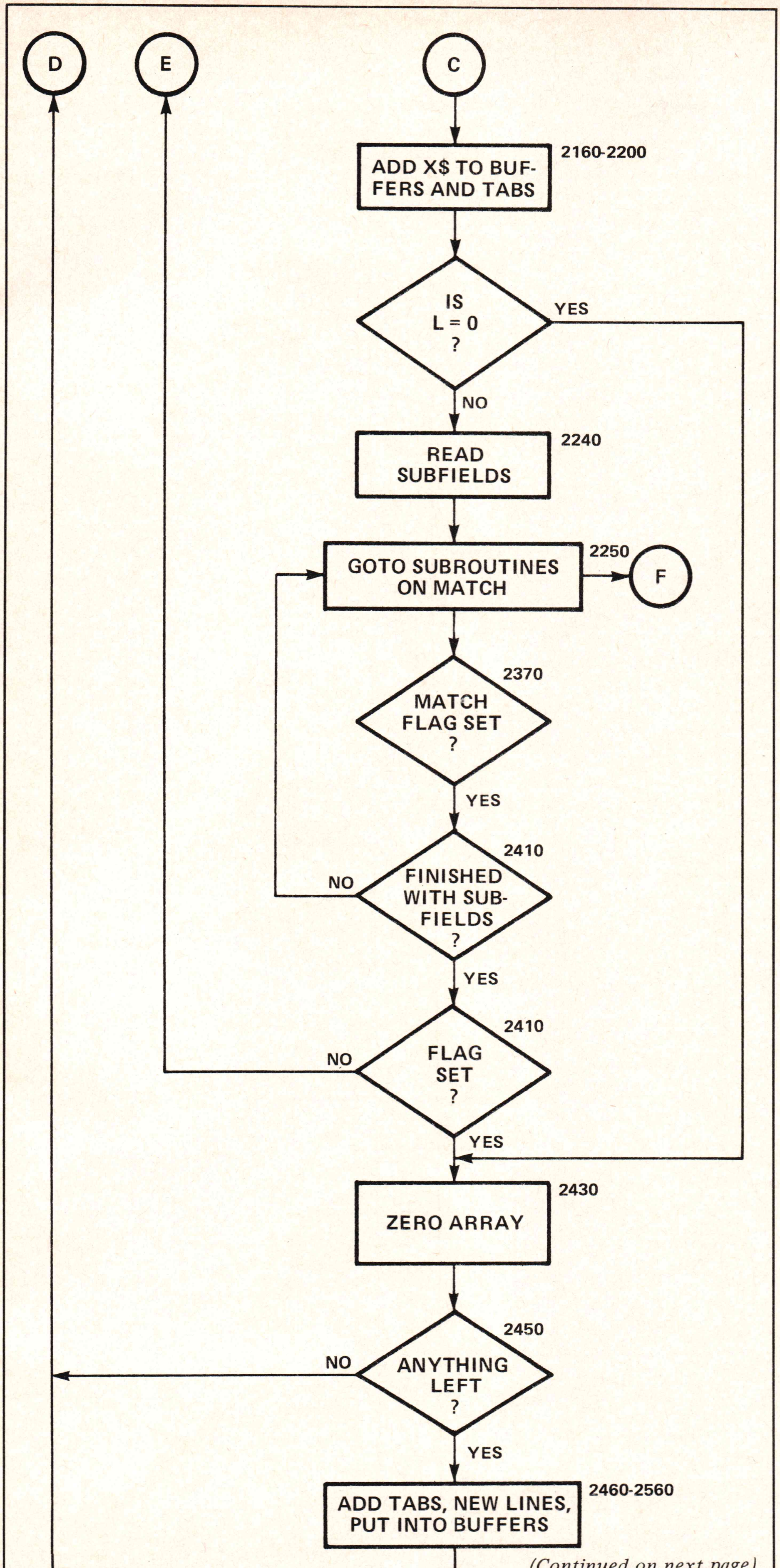
Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 241**



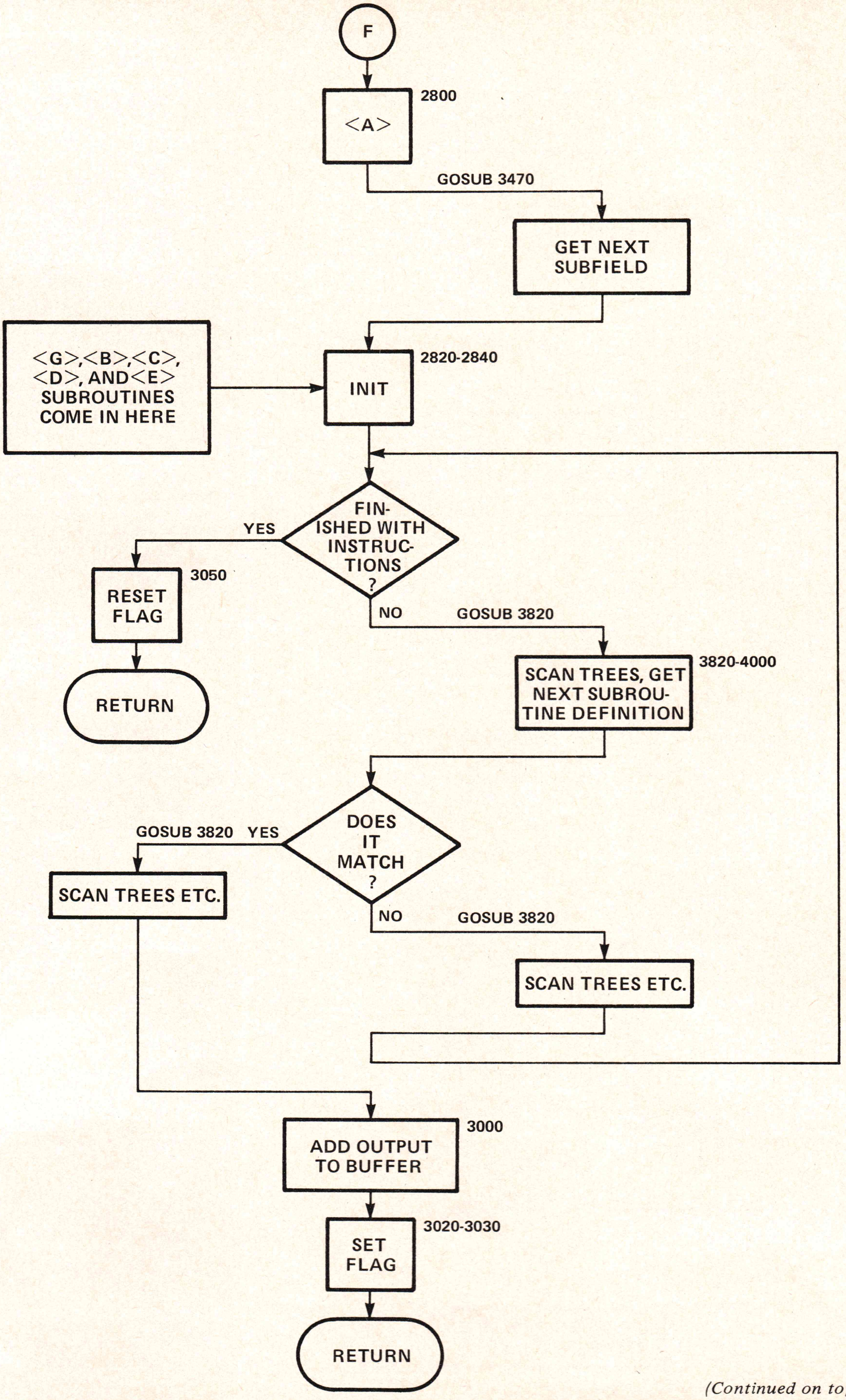
(Continued on next page)



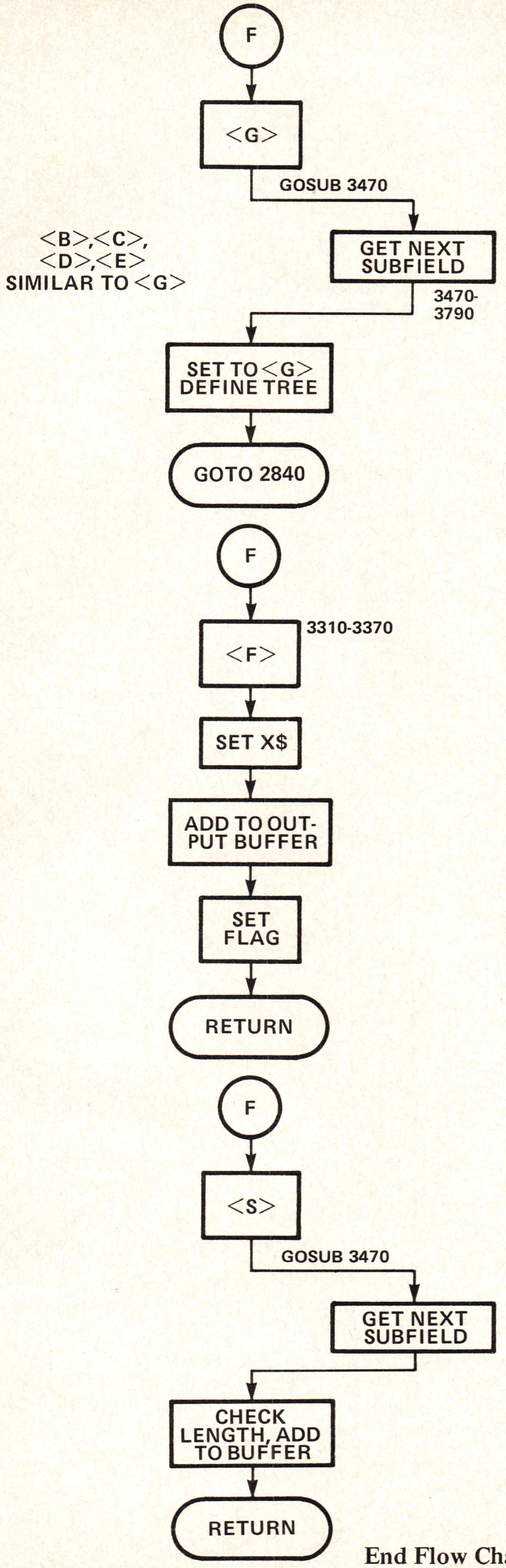
(Continued on top of page 65)



(Continued on next page)



(Continued on top of page 67)



End Flow Chart

8080 to Z80 Translator

(Text begins on page 60)

```
10 '8080 TO Z80 TRANSLATOR
20 '   FOR USE ON TRS-80, MODEL I, SINGLE DISK SYSTEM
30 '   USES ULTRADOS OPERATING SYSTEM
40 '   CREATES Z80 SOURCE FILE FOR EDITOR/ASSEMBLER
50 '   ORIGINAL VERSION BY ROBERT W. DEA, DDJ MAY80 #45 P.48
60 '   MODIFIED VERSION BY ANTHONY T. SCARPELLI, JUNE 1982
70 '
80 '
90 '   THE DATA IN THE ARRAY ARE DEFINED AS FOLLOWS:
100 '       L.8080MN.N.Z80MN.SUB1.SUB2....
110 ' L       = NUMBER OF CHARACTERS IN 8080 MNEMONIC
120 ' 8080MN  = 8080 INSTRUCTION
130 ' N       = NUMBER OF SUBFIELDS
140 ' Z80MN   = Z80 EQUIVALENT INSTRUCTION
150 ' SUBN    = SUBFIELD DEFINITIONS
160 '
170 ' THE SUBFIELD DEFINITIONS ARE
180 '    <A> - REGULAR ONE TO ONE REGISTER TRANSLATION
190 '    <G> - RST NUMBER TRANSLATION
200 '    <B> - REGISTER PAIR TRANSLATION
210 '    <C> - REGISTER PAIR TO ACCUMULATOR
220 '    <D> - ACCUMULATOR TO REGISTER PAIR
230 '    <E> - EDIT A COMMA
240 '    <F> - EDIT FOLLOWING CHARACTER STRING TO OUTPUT
250 '    <S> - ONE TO ONE CHARACTER STRING TRANSLATION
260 '
270 '
280 '*** NOTE: THIS VERSION USES A DOUBLE SPEED MODIFICATION
290 '          OUT 254,1 CAUSES DOUBLE SPEED CHANGEOVER
300 '          OUT 254,0 CAUSES NORMAL SPEED CHANGEOVER
310 '
320 '
330 CLS:PRINT"     **** 8080 TO Z80 TRANSLATOR ****"
340 PRINT:PRINT"ENTER 'C' TO CREATE AN 8080 SOURCE FILE"
350 PRINT"ENTER 'T' TO TRANSLATE 8080 TO Z80 CODE AND"
360 PRINT"          CREATE EDTASM SOURCE FILE"
370 PRINT"ENTER 'X' TO EXIT PROGRAM"
380 INPUT Q$
390 IF Q$="C" THEN 4820
400 IF Q$="X" THEN OUT 254,0:END
410 IF Q$<>"T" THEN 340
420 '
430 CLEAR 0:CLEAR 1500:DIM DA$(83)
440 ON ERROR GOTO 5320
450 ' MAKE SURE TWO BUFFERS ARE AVAILABLE (ULTRADOS)
460 IF PEEK(&521A)<2 THEN CMD"O":GOTO 460
470 '
480  DA$(1)="3.ACI.3.ADC.<F>.A,.<S>"
490  DA$(2)="3.ADC.3.ADC.<F>.A,.<A>"
500  DA$(3)="3.ADD.3.ADD.<F>.A,.<A>"
510  DA$(4)="3.ADI.3.ADD.<F>.A,.<S>"
520  DA$(5)="3.ANA.1.AND.<A>"
530  DA$(6)="3.ANI.1.AND.<S>"
540  DA$(7)="4.CALL.1.CALL.<S>"
550  DA$(8)="2.CC.3.CALL.<F>.C,.<S>"
560  DA$(9)="2.CM.3.CALL.<F>.M,.<S>"
570  DA$(10)="3.CMA.0.CPL"
580  DA$(11)="3.CMC.0.CCF"
590  DA$(12)="3.CMP.1.CP.<A>"
600  DA$(13)="3.CNC.3.CALL.<F>.NC,.<S>"
610  DA$(14)="3.CNZ.3.CALL.<F>.NZ,.<S>"
620  DA$(15)="2.CP.3.CALL.<F>.P,.<S>"
630  DA$(16)="3.CPE.3.CALL.<F>.PE,.<S>"
640  DA$(17)="3.CPI.1.CP.<S>"
650  DA$(18)="3.CPO.3.CALL.<F>.PO,.<S>"
660  DA$(19)="2.CZ.3.CALL.<F>.Z,.<S>"
670  DA$(20)="3.DAA.0.DAA"
680  DA$(21)="3.DAD.3.ADD.<F>.HL,.<B>"
690  DA$(22)="2.DB.1.DEFB.<S>"
700  DA$(23)="3.DCR.1.DEC.<A>"
710  DA$(24)="3.DCX.1.DEC.<B>"
720  DA$(25)="2.DI.0.DI"
730  DA$(26)="2.DM.1.DEFM.<S>"
```

(Continued on page 70)

8080 to Z80 Translator

(Listing continued, text begins on page 60)

```
740  DA$(27)="2.DW.1.DEFW.<S>"
750  DA$(28)="2.EI.Ø.EI"
760  DA$(29)="3.EQU.1.EQU.<S>"
770  DA$(3Ø)="3.HLT.Ø.HALT"
780  DA$(31)="2.IN.5.IN.<F>.A,(.<S>.<F>.)."
790  DA$(32)="3.INR.1.INC.<A>"
800  DA$(33)="3.INX.1.INC.<B>"
810  DA$(34)="2.JC.3.JP.<F>.C,.<S>"
820  DA$(35)="2.JM.3.JP.<F>.M,.<S>"
830  DA$(36)="3.JMP.1.JP.<S>"
840  DA$(37)="3.JNC.3.JP.<F>.NC,.<S>"
850  DA$(38)="3.JNZ.3.JP.<F>.NZ,.<S>"
860  DA$(39)="2.JP.3.JP.<F>.P,.<S>"
870  DA$(4Ø)="3.JPE.3.JP.<F>.PE,.<S>"
880  DA$(41)="3.JPO.3.JP.<F>.PO,.<S>"
890  DA$(42)="2.JZ.3.JP.<F>.Z,.<S>"
900  DA$(43)="3.LDA.5.LD.<F>.A,(.<S>.<F>.)"
910  DA$(44)="4.LDAX.1.LD.<D>"
920  DA$(45)="4.LHLD.5.LD.<F>.HL,(.<S>.<F>.)"
930  DA$(46)="3.LXI.3.LD.<B>.<E>.<S>"
940  DA$(47)="3.MOV.3.LD.<A>.<E>.<A>"
950  DA$(48)="3.MVI.3.LD.<A>.<E>.<S>"
960  DA$(49)="3.NOP.Ø.NOP"
970  DA$(5Ø)="3.ORA.1.OR.<A>"
980  DA$(51)="3.ORG.1.ORG.<S>"
990  DA$(52)="3.ORI.1.OR.<S>"
1000  DA$(53)="3.OUT.5.OUT.<F>.(.<S>.<F>.),A"
1010  DA$(54)="4.PCHL.2.JP.<F>.(HL)"
1020  DA$(55)="3.POP.1.POP.<B>"
1030  DA$(56)="4.PUSH.1.PUSH.<B>"
1040  DA$(57)="3.RAL.Ø.RLA"
1050  DA$(58)="3.RAR.Ø.RRA"
1060  DA$(59)="2.RC.2.RET.<F>.C"
1070  DA$(6Ø)="3.RET.Ø.RET"
1080  DA$(61)="3.RLC.Ø.RLCA"
1090  DA$(62)="2.RM.2.RET.<F>.M"
1100  DA$(63)="3.RNC.2.RET.<F>.NC"
1110  DA$(64)="3.RNZ.2.RET.<F>.NZ"
1120  DA$(65)="2.RP.2.RET.<F>.P"
1130  DA$(66)="3.RPE.2.RET.<F>.PE"
1140  DA$(67)="3.RPO.2.RET.<F>.PO"
1150  DA$(68)="3.RRC.Ø.RRCA"
1160  DA$(69)="3.RST.1.RST.<S>"
1170  DA$(7Ø)="2.RZ.2.RET.<F>.Z"
1180  DA$(71)="3.SBB.3.SBC.<F>.A,.<A>"
1190  DA$(72)="3.SBI.3.SBC.<F>.A,.<S>"
1200  DA$(73)="4.SHLD.5.LD.<F>.(.<S>.<F>.),HL"
1210  DA$(74)="4.SPHL.2.LD.<F>.SP,HL"
1220  DA$(75)="3.STA.5.LD.<F>.(.<S>.<F>.),A"
1230  DA$(76)="4.STAX.1.LD.<C>"
1240  DA$(77)="3.STC.Ø.SCF"
1250  DA$(78)="3.SUB.1.SUB.<A>"
1260  DA$(79)="3.SUI.1.SUB.<S>"
1270  DA$(8Ø)="4.XCHG.2.EX.<F>.DE,HL"
1280  DA$(81)="3.XRA.1.XOR.<A>"
1290  DA$(82)="3.XRI.1.XOR.<S>"
1300  DA$(83)="4.XTHL.2.EX.<F>.(SP),HL"
1310  '
1320  ' INITIALIZE SYNTAX TREE STRINGS
1330  A3$="A A B B C C D D E E H H L L M (HL)"
1340  B$="PSW AF B BC D DE H HL SP SP"
1350  C$="B (BC),A D (DE),A"
1360  D$="B A,(BC) D A,(DE)"
1370  E$=" , ,"
1380  G$="Ø Ø 1 8 2 1ØH 3 18H 4 2ØH 5 28H 6 3ØH
      7 38H"
1390  '
1400  ' CREATE FILE NAMES AND BUFFERS
1410  PRINT:INPUT"ENTER 8080 FILE NAME ";I9$
1420  OUT 254,Ø:  'NORMAL SPEED
1430  OPEN "R",1,I9$
1440  FIELD 1, 255 AS A$
1450  INPUT"ENTER Z80 DISK FILESPEC ";F9$
```

```
1460 INPUT"ENTER EDTASM SOURCE FILE NAME (6 CHAR. MAX) ";O9$
1470 IF LEN(O9$)>6 THEN PRINT"** TOO LONG **":GOTO 1460
1480 OPEN "R",2,F9$
1490 FIELD 2, 255 AS Z$
1500 OUT 254,1   ' *** DOUBLE SPEED
1510 R=1:F1=1:W=1:C9=0
1520 '
1530 ' FILL IN FILE NAME WITH SPACES IF NECESSARY
1540 K=LEN(O9$)
1550 IF K=6 THEN 1590
1560 FOR I=K TO 5
1570   O9$=O9$+" "
1580 NEXT I
1590 O1$=CHR$(211)+O9$
1600 '
1610 ' GENERATE AUTO LINE NUMBERING & INC BY 10
1620 INPUT"START LINE NUMBERS AT ";C0:PRINT
1630 C0=C0-10:C1=900000+C0
1640 ' INC LINE COUNT BY 10
1650 C1=C1+10:C3$=""
1660 ' CONVERT TO ASCII NUMBER 5 PLACES
1670 C1$=STR$(C1)
1680 C1$=RIGHT$(C1$,5)+" "
1690 ' CONVERT TO TOKENS
1700 FOR I=1 TO 5
1710   C4$=CHR$(VAL(MID$(C1$,I,1))+176)
1720   C3$=C3$+C4$
1730 NEXT I
1740 C2$=C1$
1750 C3$=C3$+" "
1760 PRINT"- ";C2$+" - ";
1770 '
1780 ' GET NEXT 8080 INSTRUCTION
1790 Z1$=CHR$(91)
1800 R1$=""
1810 IF F1>C9 THEN 4510
1820 R1$=R1$+MID$(R$,F1,1)
1830 IF RIGHT$(R1$,1)=Z1$ THEN 1860
1840 F1=F1+1
1850 GOTO 1810
1860 F1=F1+1
1870 I$=LEFT$(R1$,LEN(R1$)-1)
1880 PRINT I$
1890 '
1900 ' DO WE HAVE END OF TRANSLATION?
1910 IF I$<>"@EXIT" THEN 1980
1920 PRINT:PRINT"IF NOT END OF LISTING NOTE LINE NUMBER."
1930 INPUT"PRESS <ENTER> TO CLOSE FILE AND RETURN TO MENU";X$
1940 GOTO 4620
1950 '
1960 ' EDIT LINE# IN AND CONVERT TO Z80 INSTRUCTION
1970 ' CHECK FOR JUST COMMENT LINE
1980 IF LEFT$(I$,1)<>";"THEN 2010
1990 O$=O$+I$:O2$=O2$+I$
2000 GOTO 2660
2010 GOSUB 4030
2020 '
2030 ' DO WE HAVE AN 8080 INSTRUCTION IN INSTRUCTION BUFFER?
2040  IF LEN(N$)=0 THEN 2660
2050 ' YES! NOW SEARCH FOR MATCH
2060  GOSUB 4270
2070 '
2080 ' THERE WAS A MATCH, Z8$=8080 CODE
2090 '   GET NEXT STRING
2100 ' INIT OUTPUT STRING BUFFER
2110  O$="":O2$=""
2120  GOSUB 4030
2130 '
2140 ' NOW GET Z80 OUTPUT INSTR AND EDIT TO OUTPUT BUFFERS
2150 GOSUB 4380
2160 O$=O$+X$:O2$=O2$+X$
2170 FOR K=LEN(O2$)+1 TO 16
2180   O2$=O2$+" "
2190 NEXT K
2200 O$=O$+CHR$(09)

2210 '
2220 ' NOW READ TRANSLATION SUBFIELDS
2230 IF L=0 THEN 2430
```

(Continued on next page)

8080 to Z80 Translator

(Listing continued, text begins on page 60)

```
2240 FOR I=2 TO L+1
2250    S$=S1$(I)
2260    ' CHECK SUBSET FOR DEFINITION MATCH AND TRANSFER TO
2270    '    DEFINITION PROCESS SECTION.
2280     IF S$="<A>" THEN GOSUB 2800
2290     IF S$="<G>" THEN GOSUB 3100
2300     IF S$="<B>" THEN GOSUB 3150
2310     IF S$="<C>" THEN GOSUB 3190
2320     IF S$="<D>" THEN GOSUB 3230
2330     IF S$="<E>" THEN GOSUB 3270
2340     IF S$="<F>" THEN GOSUB 3310
2350     IF S$="<S>" THEN GOSUB 3390
2360    ' IS MATCH FLAG SET?
2370     IF F=1 THEN 2410
2380    ' WE GET HERE IF PARSE FAILED
2390    GOTO 2110
2400    ' YES WE HAVE A GOOD TRANSLATION
2410    NEXT I:IF F<>1 THEN 2110
2420 ' ZERO ARRAY
2430 FOR I=1 TO 6:S1$(I)="":NEXT I
2440 ' EDIT ANYTHING THATS LEFT
2450  IF T+1>LEN(I$) THEN 2560
2460  FOR K=LEN(O2$)+1 TO 29
2470    O2$=O2$+" "
2480  NEXT K
2490  O$=O$+CHR$(09)+CHR$(09)
2500  FOR K=1 TO LEN(I$)
2510    T1$=MID$(I$,K,1)
2520    IF T1$=";" THEN GOTO 2540
2530  NEXT K
2540 O$=O$+RIGHT$(I$,LEN(I$)-K+1)
2550 O2$=O2$+RIGHT$(I$,LEN(I$)-K+1)
2560  GOTO 2660
2570 '
2580 ' NO FIND EXIT
2590 O2$="*NO MATCH* "+I$
2600 ' OUTPUT *NO MATCH* MSG AND DEC LINE COUNTER BY 10
2610 PRINT O2$
2620 C1=C1-10
2630 O$="":GOTO 2690
2640 '
2650 ' EDIT LINE# IN AND OUTPUT THE TRANSLATED Z80 INST
2660  O$=C3$+O$
2670  PRINT C1$+O2$
2680  O$=O$+CHR$(13)
2690  L9=LEN(O1$)+LEN(O$)
2700  IF L9<255 THEN 2750
2710  L8=255-LEN(O1$)
2720  O1$=O1$+LEFT$(O$,L8)
2730  O$=RIGHT$(O$,LEN(O$)-L8)
2740  GOTO 4710
2750  O1$=O1$+O$
2760  O$="":O2$=""
2770  GOTO 1650
2780 '
2790 ' PROCESS SINGLE REG SECTION "<A>"
2800 GOSUB 3470
2810 ' INIT PROCESS SECTION AND SET TO <A> TREE
2820 H$=A3$
2830 ' INIT TREE SEARCH SECTION
2840 H2=0:T2=1
2850 ' CHECK IF WE PARSED THE WHOLE INPUT 8080 INST YET.
2860 ' SEE IF IT SATISFIES THE 8080 DEFINITION
2870 IF T2>LEN(H$) THEN 3050
2880 ' GO SCAN THE DEFINITION TREE FROM LEFT TO RIGHT TO
2890 ' GET NEXT SUB DEFINITION
2900 GOSUB 3820
2910 ' DOES IT MATCH THE SUB ELEMENT WE HAVE?
2920 IF M$=N$ THEN 2990
2930 ' NO IT DOES NOT, SO SKIP THE NEXT Z80 SUB TRANSLATION
2940 '    AND GO TRY FOR ANOTHER 8080 SUB DEFINITION MATCH.
2950 GOSUB 3820
2960 GOTO 2870
```

```
2970 ' WE HAVE AN 8080 SUB DEFINITION MATCH. NOW GET THE
2980 '    Z80 TRANSLATION AND EDIT IT INTO THE OUTPUT BUFFER O$.
2990 GOSUB 3820
3000 O$=O$+M$:O2$=O2$+M$
3010 ' SET MATCH FLAG
3020 F=1
3030 RETURN
3040 ' SET NO MATCH FLAG
3050 F=0
3060 RETURN
3070 '
3080 ' PROCESS "<G>" DEFINITIONS
3090 ' GET NEXT INPUT SUB FIELD
3100 GOSUB 3470
3110 ' SET TO <G> DEFINITION TREE
3120 H$=G$
3130 GOTO 2840
3140 ' PROCESS "<B>" DEF
3150 GOSUB 3470
3160 H$=B$
3170 GOTO 2840
3180 ' PROCESS "<C>" DEF
3190 GOSUB 3470
3200 H$=C$
3210 GOTO 2840
3220 ' PROCESS "<D>" DEF
3230 GOSUB 3470
3240 H$=D$
3250 GOTO 2840
3260 ' PROCESS "<E>" DEF
3270 GOSUB 3470
3280 H$=E$

3290 GOTO 2840
3300 ' PROCESS "<F>" DEF
3310 X$=S1$(I+1)
3320 ' JUST EDIT NEXT Z80 TRANS ELEMENT TO OUTPUT BUFFER
3330 O$=O$+X$:O2$=O2$+X$
3340 ' SET MATCH FLAG
3350 F=1
3360 I=I+1
3370 RETURN
3380 ' PROCESS "<S>" DEF
3390 GOSUB 3470
3400 F=1
3410 IF LEN(N$)=0 THEN F=0
3420 O$=O$+N$:O2$=O2$+N$
3430 RETURN
3440 '
3450 ' SCANNER RETURNS N$
3460 ' INIT SUB FIELD SCANNER SECTION
3470 H=T+1
3480 N$=""
3490 H1=H
3500 ' SCAN FOR NEXT 8080 SUB FIELD STARTING WITH FIRST
3510 '    NON-BLANK.
3520 FOR J=H1 TO LEN(I$)
3530    ' GET NEXT CHAR
3540    ' IS IT A BLANK?
3550    IF MID$(I$,J,1)<>" " THEN J=LEN(I$):GOTO 3590
3560    ' YES! GO GET NEXT CHAR
3570    H=H+1
3580    T=T+1
3590 NEXT J
3600 ' WE SHOULD BE POINTING TO START OF 8080 SUB FIELD
3610 H1=H
3620 ' SCAN UNTIL WE GET A TERMINATOR OR END OF LINE.
3630 ' ("BLANK" AND "," ARE TERMINATORS)
3640 FOR J=H1 TO LEN(I$)
3650    ' GET NEXT CHAR
3660    ' IS IT A BLANK?
3670    IF MID$(I$,J,1)=" " THEN GOTO 3790
3680    ' IS IT A ","
3690    IF MID$(I$,J,1)="," THEN GOTO 3770
3700    ' NO! ITS NONE OF THE ABOVE SO EDIT THIS CHAR INTO
3710    '    INSTRUCTION BUFFER.
3720    N$=N$+MID$(I$,J,1)
3730    T=T+1
3740    ' GO GET NEXT CHAR
```

(Continued on next page)

8080 to Z80 Translator

(Listing continued, text begins on page 60)

```
3750 NEXT J
3760 RETURN
3770 IF LEN(N$)=0 THEN T=T+1
3780 IF LEN(N$)=0 THEN N$=","
3790 RETURN
3800 '
3810 ' SCAN THE TREES
3820 H2=T2
3830 M$=""
3840 H3=H2
3850 FOR J=H3 TO LEN(H$)
3860   ' TAB TO NEXT NON-BLANK CHAR
3870   IF MID$(H$,J,1)<>" "THEN J=LEN(H$):GOTO 3900
3880   H2=H2+1
3890   T2=T2+1
3900 NEXT J
3910 H3=H2
3920 ' GET NEXT SUB DEF OF 8080 INSTRUC.
3930 FOR J=H3 TO LEN(H$)
3940   ' HAVE WE A BLANK YET?
3950   IF MID$(H$,J,1)=" " THEN J=LEN(H$):GOTO 3990
3960   ' NO! THEN ADD THIS CHAR
3970   M$=M$+MID$(H$,J,1)
3980   T2=T2+1
3990 NEXT J
4000 RETURN
4010 '
4020 ' SECTION TO EDIT A LABEL AND OUTPUT Z80 TRANSLATION
4030 H=0:T=1
4040 ' IS THERE A CHAR IN COL#1?
4050 IF LEFT$(I$,1)=" " THEN 4170
4060 H=1
4070 FOR K=1 TO LEN(I$)
4080   ' GET NEXT CHAR
4090   ' IS IT A BLANK
4100   IF MID$(I$,K,1)=" " THEN K=LEN(I$):GOTO 4150
4110   ' NO! NOT A BLANK THEN MUST BE PART OF LABEL FIELD.
4120   ' SO EDIT IT INTO OUTPUT BUFFER
4130   O$=O$+MID$(I$,K,1):O2$=O2$+MID$(I$,K,1)
4140   T=K+1
4150 NEXT K
4160 ' TAB TO COL 8 FOR PRINTOUT
4170 FOR K=LEN(O2$)+1 TO 8
4180   O2$=O2$+" "
4190 NEXT K
4200 ' INSERT TAB INTO OUTPUT BUFFER
4210 O$=O$+CHR$(09)
4220 ' NOW SCAN FOR THE 8080 INSTRUC.
4230 GOSUB 3470
4240 RETURN
4250 '
4260 ' BINARY SEARCH THROUGH DATA STATEMENTS FOR MATCH
4270 U=83:L=1
4280 F=INT((L+U)/2)   ' CALCULATE FENCE
4290 N=VAL(LEFT$(DA$(F),1)):Z8$=MID$(DA$(F),3,N)
4300 IF Z8$=N$ THEN RETURN
4310 IF L>=U THEN 2590
4320 IF Z8$>N$ THEN 4340
4330 L=F+1:GOTO 4280
4340 U=F-1:GOTO 4280
4350 GOTO 2590   ' NO MATCH
4360 '
4370 ' SUBROUTINE TO READ THROUGH ARRAY DATA
4380 L=VAL(LEFT$(DA$(F),1)):N1=L+4
4390 L=VAL(MID$(DA$(F),N1,1))
4400 FOR I=1 TO L+1
4410   FOR J=1 TO 6
4420     S1$=MID$(DA$(F),N1+1+J,1)
4430     IF S1$="." THEN N1=N1+J:J=6:GOTO 4450
4440     S1$(I)=S1$(I)+S1$
4450   NEXT J
4460 NEXT I
4470 X$=S1$(1)
```

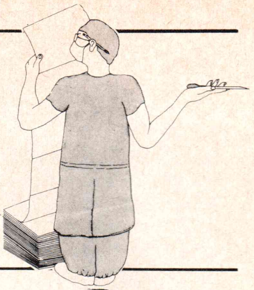
```
4480 RETURN
4490 '
4500 ' GET RECORD FROM DISC
4510 R$=""
4520 OUT 254,0
4530 GET 1,R
4540 R$=A$
4550 F1=1
4560 R=R+1
4570 C9=LEN(R$)
4580 OUT 254,1    '*** DOUBLE SPEED
4590 GOTO 1800
4600 '
4610 ' CLOSE FILES
4620 IF LEN(O1$)<0 THEN OUT 254,0:GOTO 4670
4630 O1$=O1$+CHR$(26)
4640 LSET Z$=O1$
4650 OUT 254,0
4660 PUT 2,W
4670 CLOSE 1:CLOSE 2
4680 GOTO 330
4690 '
4700 ' PUT RECORD ONTO DISC
4710 LSET Z$=O1$
4720 OUT 254,0
4730 PUT 2,W
4740 W=W+1
4750 O1$=""
4760 'OUT 254,1
4770 GOTO 2750
4780 '
4790 '
4800 '
4810 ' SECTION TO CREATE AN 8080 SOURCE FILE
4820 '
4830 CLEAR 0:CLEAR 1500:CMD"O"   ' MAKE SURE BUFFER AVAILABLE
4840 R=1:A$=""
4850 '
4860 CLS:PRINT"      *  8080 CREATE PROGRAM  *"
4870 PRINT
4880 PRINT"TYPE 8080 SOURCE IN FOLLOWING FORMAT:"
4890 PRINT"  LABEL - OPERATION - OPERAND - COMMENT"
4900 PRINT"  EXAMPLE:  START LXI H,0001H ;LOAD ADDRESS
4910 PRINT"                XRA A ;CLEAR A REGISTER"
4920 PRINT"  ONLY ONE SPACE IS REQUIRED BETWEEN ENTRIES."
4930 PRINT"A LABEL OR LEADING ';' SHOULD NOT START WITH"
4940 PRINT"  A SPACE, AN OPERATION SHOULD."
4950 PRINT"AN 'N' AFTER THE 'SURE' WILL CANCEL ENTRY,"
4960 PRINT"  AN <ENTER> ADDS ENTRY TO FILE."
4970 PRINT"TYPE '@EXIT' TO EXIT TO MENU."
4980 PRINT"A BUFFER FROM 215 TO 255 CHARACTERS LONG IS"
4990 PRINT"  SAVED TO DISK."
5000 PRINT:INPUT"ENTER 8080 OUTPUT FILE NAME ";R$
5010 PRINT"OUTPUT FILE '";R$;"' IS ON DRIVE #0"
5020 A3$="@EXIT"+CHR$(91)
5030 OPEN "R",1,R$
5040 FIELD 1,255 AS A$
5050 PRINT
5060 X$=""
5070 PRINT"     ";CHR$(92)
5080 LINE INPUT"--> "; A1$
5090 B=LEN(A1$)+1:C=LEN(A2$)
5100 IF B+C>255 THEN PRINT"NEXT ENTRY WILL EXCEED
 BUFFER SPACE AND PRODUCE ERROR!!   SAY 'N'!
 YOU HAVE ONLY";255-LEN(A2$);" CHARACTERS LEFT!"
5110 INPUT"SURE";X$
5120 IF X$="N" THEN 5060
5130 A1$=A1$+CHR$(91)
5140 A=LEN(A2$)
5150 IF A>215 THEN 5190
5160 A2$=A2$+A1$
5170 IF A1$=A3$ THEN 5200
5180 GOTO 5060
5190 IF LEN(A1$)<1 THEN 5240
5200 LSET A$=A2$
5210 PUT 1,R
5220 R=R+1
5230 IF A1$=A3$ THEN 5270 ELSE 5250
5240 A1$=""
```

(Continued at right, top of page)

```
5250 A2$=A1$
5260 GOTO 5060
5270 CLOSE 1
5280 GOTO 330
5290 END
5300 '
5310 ' ERROR TRAP
5320 OUT 254,0
5330 RESUME
5340 '
5350 ' END OF PROGRAM
```

End Listing

DR. DOBB'S CLINIC



by D. E. Cortesi

Disk Drives Us Crazy

In February we presented Loren Amelang's request for explanations why perfectly good diskettes could come up with read errors after being stored for a while. David Oster (whose letterhead proclaims The People's Republic of Santa Monica, its motto "Unity and Breakfast for All") suggests two ways. First, he points out, drives vary with time. "Apple drives, in particular, seem to need frequent realignment. The disks you read and write every day change with the drive, but the drive changes enough that stored disks stop working."

Amelang also wondered why one program could always read the bad diskettes when others couldn't. Oster has a comment on this, too.

"When I had this problem it turned out that the programs were accessing the disk in very different ways. When the program that failed wanted to go to an inner track like track 70, it would tell the controller "go to track 70." When the program that worked wanted to go there, it told the controller "go to the next track," 70 times. If you tell a disk controller to step once, it just does it. If you tell it to seek a long way, it runs an internal subroutine that does a fancy acceleration-deceleration of the head carriage. Only when it gets there does it begin looking for a track. If the disk is a slightly different shape than the controller expects, the position error in going 70 tracks is 70 times as large as the position error from stepping one track.

"I had this problem because my drives weren't properly cooled. There was a nice big fan in the cabinet, but there were a couple of unused connector holes and all the air flowed through them and not enough of it flowed over the electronics. I replaced a few heat-damaged chips, covered the extra holes with tape, and haven't had a problem since."

OK, those are reasonable propositions (especially the part about air flow), but they sound to us as if they would be hardware-specific. For example, according to its data sheets, the Western Digital FD1793 controller (used in the popular California Computer Systems 2422 board) performs a seek from track zero to track 70 by doing 70 "step-in" operations. And the 1793 doesn't make any adjustments on the resulting position; it just assumes that a step is a step.

There must be a lot more things that could be said about Loren Amelang's problem. Another open problem is that of Ernest Knipp, whose Z80 system occasionally goes catatonic. We look forward to more contributions on both these topics.

Counts and Measures

In that February issue, we gave Burks Smith room to speak out on the use of floating-point arithmetic, and its lack of precision as usually implemented. The heart of his presentation may have been his statement that "If something called 'double-precision arithmetic' can't take ten percent of a dollar and get *exactly* ten cents, there shouldn't be any excuses. The answer is just plain wrong!" Oster has a comment on that.

"Smith has confused *counts* with *measures*. When you say you have ten dollars, you are saying you have 1,000 pennies – a count. When you say you have ten gallons of water, you are not saying that you have a certain number of water molecules, but making a statement about a physical quantity that you only know to a certain precision – a measure. Floating-point numbers were designed for calculation with measures; integers were designed for calculation with counts.

"So, if something called 'double-precision arithmetic' takes ten percent of a dollar, it is right as long as its answer lies between 0.999. . . 9 and 1.000. . . 01. It may not be what you *want*, but it is what that arithmetic is designed to do. There are some experimental computer languages where all numbers input to a program must be labeled with whether they are measures or counts, and if measures, how accurate they are and what units they are in. The results of the program are automatically expressed in appropriate units and you are told how precise the results are."

Distinguishing "counts" and "measures" certainly helps clarify the problem in our mind, at least. On reflection, it seems that the distinction we try to draw between "commercial" and "scientific" applications is based on which kind of numbers is preponderant in each. What we call "commercial" work deals mostly with counts; "scientific" computing deals mostly with measures. Think about it.

But there is more to be said on this topic as well, and your contribution is still welcome. We asked other questions in that February column. CBASIC uses a BCD representation for float numbers that has about the same precision as MBASIC's binary one – has anyone really compared the results of the two on the same problem set? We also wondered if there were any ratios that have a finite representation in binary but not in decimal (as there are decimal fractions that can't be represented in binary). Oster's answer is "no"; can you figure out what leads him to this conclusion?

Spotting DDT

Back in October of 1982, we showed one way to get DDT to load in such a way that the resident parts of CP/M were still visible and the CCP wasn't overlaid. Aubrey Hutchison has another way. If you know where you want DDT to load itself, the most direct way is to alter its code. Use DDT to modify DDT.COM. Look at the instructions around 0150h; you should find

```
0150 SUB B
0151 MOV D,A
```

If you replace those two instructions with the single instruction

0150 MVI D,*xx*

you will have a version of DDT that will always load itself at *xx*00h in storage.

Fun With Your New BIOS

We're working at bringing up CP/M 3 (or "CP/M Plus," as Digital Research wants to call it) on our own system. The first step is to get the present BIOS to assemble with a relocating assembler. We're using RMAC and LINK80 from Digital Research, mostly because RMAC allows the long, long labels that we like to use.

Rather than do a whole lot of new code, then debug it while learning a new operating system, we thought we would first make the new BIOS work with CP/M 2.2. Then we'd enhance it, still on 2.2, and finally make the few changes needed for CP/M 3. One of the enhancements we want to make while still running the old operating system is to split the BIOS into a fixed part and a banked part. The more complicated disk functions go into the banked BIOS; the serial I/O and the trivial disk functions like "set track" remain in the fixed part.

With the complicated functions set aside in banked storage, we can really let them spread out and get competent. For example, we plan to display disk error messages in detail, in English. That's something there just isn't room for when the whole BIOS has to be squeezed into a

couple of kilobytes. Another planned enhancement is the use of separate read and write sector buffers for each physical drive, so when programs alternate reading and writing they won't force the same big sectors to be read over and over. Still another is the mapping of four logical drives onto two physical ones, as is done in the IBM PC. Doing this and maintaining the full generality of use for all four drives turns out to be trickier than we thought, but it's possible when a few hundred additional bytes don't matter.

Fun With LINK80

OK, so you plan a much larger BIOS. Obviously you don't want to keep it as one big assembly; it's too difficult to edit and too tedious to assemble that way. You want the BIOS broken into lots of small modules, each with a well-defined set of functions (we ended up with a dozen, five fixed and seven banked). It's one of the great advantages of having a linker that you can do that easily, right?

Wrong. It isn't easy at all with LINK-80, because a two-part BIOS is not the sort of program it was designed to link. It doesn't help that the manual addresses the use of the program only as an adjunct to PL/I-80 and doesn't explain it as a general tool. We managed it in the end and these are some of the things we learned along the way.

First, the "L" option determines the origin of the linked program *and* the starting point of the linker's output file. If you were linking a (non-banked) BIOS that ran at F200h, you would do it like this:

```
LINK80 BIOS=modules...[LF200]
```

That produces BIOS.COM, containing just the code of the given modules, linked to an origin of F200h. The same effect can be had with the "P" option, but the resulting file contains enough binary zeroes to fill storage from 0100h to the program origin – about 61K of them in the example. We can't figure out what the "P" option is good for.

Next, the "D" option determines the origin of the data segments of the linked modules, while the "L" option sets the origin of the code segments. We coded our banked BIOS so that the banked modules consist entirely of data segments and the fixed modules are entirely code segments. We casually assumed that a command like

```
LINK80 BIOS=modules...
     [D4000,LF200]
```

would link the banked parts to an origin of 4000h and the fixed parts at F200h. It doesn't. It produces a file that contains only the code segments, linked to F200h. The data segments vanish. In fact, LINK80 will never write any output that would fall below the "L" load-point, whatever segment it comes from. The linking is done correctly in that external references have correct values, but the data they point to is omitted from the file if it falls below the "L" address.

That's just as well, because we weren't thinking too clearly when we devised the command above. What we really want is two separate files, FIX.COM and BNK.COM, the first containing the code that is to go into high, global storage and the second containing code to go into the banked storage. That implies that LINK80 has to process each group of modules separately. But each section has references to public labels in the other. That implies that LINK80 has to process all modules together in one run so that it can resolve the interlocking references. If this were a movie, our computer would be saying, "Contradiction, does not compute" in a hollow voice. CP/M 3 is supposed to have a command that does all this, but we are working under CP/M 2.2.

Enter the Overlay

Just as we were about to give up, we found the appendix on overlays in the LINK80 manual. They aren't explained very well, presumably because users of PL/I-80 aren't supposed to be concerned about the details. But in fact, LINK80's overlay feature can give us just what we need for the present problem.

To LINK80, an overlay is another program, one that just happens to be linked at the same time as some main program. The syntax that specifies an overlay is identical to the syntax for a main

program, but given recursively in parentheses. The command lines could get out of hand, except that they can be continued by appending an ampersand. Here is how a PL/I user might link a program with overlays:

```
LINK80 ROOT=modules...          &
    (OV1=modules...)            &
    (OV2=modules...             &
        (OV2A=modules...)       &
        (OV2B=modules...)       )
```

That command would create five output files: ROOT.COM, OV1.OVL, OV2.OVL, OV2A.OVL, and OV2B.OVL. Each overlay file is a separate link, composed of the modules that make it up linked to an origin that is the end of its root module. OV1 and OV2 will be linked to run when loaded at the byte after ROOT.COM; OV2A and OV2B will be linked to run at the byte following OV2. All the link options apply, independently, within the parenthetical specification of an overlay.

The code in an overlay may refer to public labels in its root module, but not vice versa. That's all right; there is an escape. A more serious problem is that LINK80 insists on adding two external references to a root program, "?OVLAY" and "?OVLA0." These presumably mean something to PL/I, though we don't see why the compiler couldn't generate them in the usual way. They have to be supplied because there won't be any output unless all external references are resolved. We resolve them by writing and assembling FAKE.ASM as follows:

```
    CSEG
    PUBLIC ?OVLAY,?OVLA0
?OVLAY:
?OVLA0:
    END
```

Now we can create FIX.OVL and BNK.OVL, the two sections of our BIOS, with all external references resolved and all the code located where we want it. First, FIX.OVL:

```
LINK FAKE &
(FIX=all-12-bios-modules &
    [D4000,LF200] )
```

Because it loads at F200h, none of the bank-BIOS modules will be included in the file FIX.OVL. However, their proper addresses will be filled in where needed. The banked section is trickier:

```
LINK FAKE,all-fixed-modules &
    [LF200,$OZ] &
    (BNK=all-banked-modules &
    [L4000,$OA] )
```

Here we let LINK80 think that there is a root module, loading at F200h, which contains the fixed parts of the BIOS. The "$OZ" switch tells it not to produce any output for this "root" module. Then we tell it to make BNK.OVL, composed of our banked modules and loading at 4000h, the base of our banked BIOS. The "$OA" switch says this file is to be written. All the places in the banked code that refer to fixed code will be filled in with the proper addresses.

And that is how one treads cautiously around the pitfalls and potholes of RMAC and LINK80 to get a two-section BIOS written and linked. The result is two files that contain the code, ready to be loaded into storage in their correct locations and banks. How these parts are put together with the CCP and BDOS of CP/M 2.2 and written onto the system tracks of a diskette, and how during cold start they get loaded and located in storage, is at least two other stories. But arranging the link was the highest hurdle we had to cross.

Perhaps that wasn't the most fascinating item you've read in this column. Well, you've nobody to blame but yourself. In the absence of questions and discoveries from readers, our only source of material is the top of our desk. DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 243**

16-BIT SOFTWARE TOOLBOX

by Ray Duncan

CP/M 83 Show

The recent CP/M 83 exhibition in San Francisco drew over 45,000 attendees, a degree of success surprising even to its sponsors. Browsing through the booths, I was impressed with the very rapid shift towards 16-bit microprocessors that has become evident over the last year. Almost every new machine and application program of any significance was based on an 8086 or 68000, though Olivetti was also present, valiantly trying to rescue the Z8000 from oblivion. There was almost nothing worth mentioning in the way of new 8-bit hardware or software, with one exception: several manufacturers introduced super-miniaturized Z80 CP/M systems in which the whole system board and a slim-line 5" disk drive fit into a chassis smaller than a shoebox.

Digital Research itself displayed many new products at the show, some of which indicate marked shifts in its corporate goals. CP/M-68K was shown running on five different machines including the Sage and Compu-Pro 68000-based microcomputers. It comes with a complete set of software development tools including an assembler and a full-blown C compiler. Reportedly CP/M-68K will eventually be able to execute as a task under UNIX, giving its users the best of both worlds. The same C compiler will be available on the 8086/88 in April 1983. Digital Research also announced that all of its future operating systems products and language compilers will be written in C! Evidently DRI's previous flagship compiler, PL-1 subset G, is being relegated to the wings forever.

Also announced was a Digital Research version of "plain" CP/M-86 for the IBM Personal Computer, including printer spooling and the new graphics support nucleus GSX, for only $60.00. DRI has been discontented with IBM's half-hearted support and marketing of CP/M-86, and is now showing more convincing determination to capture some of the enormous (and affluent) PC user base from Microsoft. This is good news for PC owners, since they can now readily afford both PC-DOS and CP/M-86 and therefore be much more flexible in their application software purchases. But such a drastic drop in price will surely raise questions in the minds of CP/M owners of other machines, who must still pay $150 or more for their operating systems.

The most startling development of all was the Digital Research display of an extended LOGO language for the IBM PC. A seminar on the new product was given by Gary Kildall himself, who described it as having all the benefits of LISP but with a "user-friendly" programming environment. Life is certainly full of surprises.

Intel 8087 News

Hudson Associates has announced an 8087 piggyback board for the Godbout 8085/8088 Dual CPU. This will be a boon to those of you who wish to take advantage of the 8087 for fast floating-point math but don't want to trash your present CPU board for a new $750 Godbout 8086/87 board. The procedure for installing the piggyback board is very simple: remove the 8088 from your present CPU board, plug the piggyback board into the 8088's socket, then plug the 8088 and 8087 chips into the piggyback board. The system should then boot up and run just as before.

This piggyback board will probably also work on other 8088 CPU boards such as the Lomas LPD-88. The main consideration is whether adequate power is delivered to the 8088's socket to supply both processors. The 8087 draws considerably more power (up to 475 mA) than the 8088 (340 mA). We have used a similar hand-wired piggyback board to add an

Command	Description
AR68	Archive utility, stores object files in the C run-time library
AS68	68000 assembler
C	C language compiler
CP68	C language preprocessor for macros
DDT	Interactive 68000 debugger
DIR	Display disk file directory
DIRS	Display directory of "system" files
DUMP	Display contents of a file in hex and ASCII
ED	Line editor
ERA	Erase file(s)
LO68	Linker
NM68	Symbol table display utility
PIP	Transfer, concatenate, and/or filter files between various peripheral devices
RELOC	Relocate a command file to an absolute address
SENDC68	Convert command file to Motorola S-record format
SIZE68	Print the size of a command file

Table 1.
CP/M-68K Commands and Utilities

Offset	Contents
0000-0003	Lowest address of Transient Program Area
0004-0007	1 + highest address of TPA
0008-000B	Starting address of the Text Segment
000C-000F	Length of Text Segment (bytes)
0010-0013	Starting address of the Data Segment
0014-0017	Length of Data Segment
0018-001B	Starting address of the bss (uninitialized data)
001C-001F	Length of bss
0020-0023	Length of free memory after bss
0024-0024	Drive from which the program was loaded
0025-0037	Reserved
0038-005B	2nd parsed FCB from command line
005C-007F	1st parsed FCB from command line
0080-00FF	Command tail and default DMA buffer

Table 2.
Program Base Page Format for CP/M-68K

8087 to the NEC 8086-based Advanced Personal Computer without any problems. Hudson Associates may be contacted at P.O. Box 2957, Santa Clara, CA 95055.

A LINK-80-compatible run-time library for 8087 support has been released by Avant-Code, 1508-A Oxford Street, Berkeley, CA 94709. The library costs $200 and may be used with Fortran-80, Bascom-80, Cobol-80, or Macro-80. Of course, you need to run the linked programs on a system containing the Godbout Dual CPU, since the 8088 must take over control in order to perform the math operations on the 8087.

Preview of CP/M-68K

The information presented here is abstracted from the new CP/M-68K documentation set which was recently released. CP/M-68K is logically symmetrical to the CP/M-80 and CP/M-86 operating systems, with enhancements to support the 16-megabyte memory addressing space of the 68000 microprocessor. The disk file structure is exactly compatible with the 8080 and 8086/88 versions, and supports a maximum of 16 drives with up to 512 megabytes per drive.

The operating system resides in a file named "CPM.SYS" and is loaded into memory by a cold-start routine which is initially read in from the two reserved system tracks (similar to CP/M-86). All of the modules (CCP, BDOS, and BIOS) of CP/M-68K remain resident at all times. The CCP and BDOS are written in C, and the BIOS of course is written in assembly language by the system implementor.

CP/M-68K contains most of the familiar CP/M commands, as well as some impressive new program development tools (see Table 1). The inclusion of a version 7, C-compatible compiler is a tremendous enhancement. Preliminary benchmark results indicate that this compiler generates very efficient code (see Jim Gilbreath's article in the January 1983 *BYTE*).

Transient application programs are designated in the disk directory with the extension "68K." A program may be loaded via a command line at the CCP level or by another program through BDOS funtion 59. After a program is loaded, the Transient Program Area (TPA) contains the base page, the program segments (text, data, and bss). and the user stack (see Table 2 and Figure 1).

The CP/M-68K BDOS functions are very similar to CP/M-86, except that the memory management functions are not included. A typical calling sequence is as follows:

```
move.w #2,D0.W ;move function number
               ;to the first data register
move.w #7,D1.W ;move ASCII bell code
               ;to the second data
               :register
trap #2        ;request BDOS function
               ;to output a character
```

Any results are returned in D0.W. A few of the BDOS functions are different or new compared to the previous operating systems and will be described briefly below.

Function 12 (return version number) yields the value 2022H in register D0.W, signifying a 68000-CPU, single-user environment without networking, and BDOS version 2.2.

Function 50 is a direct BIOS call which allows application software to manipulate the primitive device drivers. The D1.L register contains the address of the BIOS Parameter Block, a five-word memory area containing the desired function number and two other 32-bit paramenters.

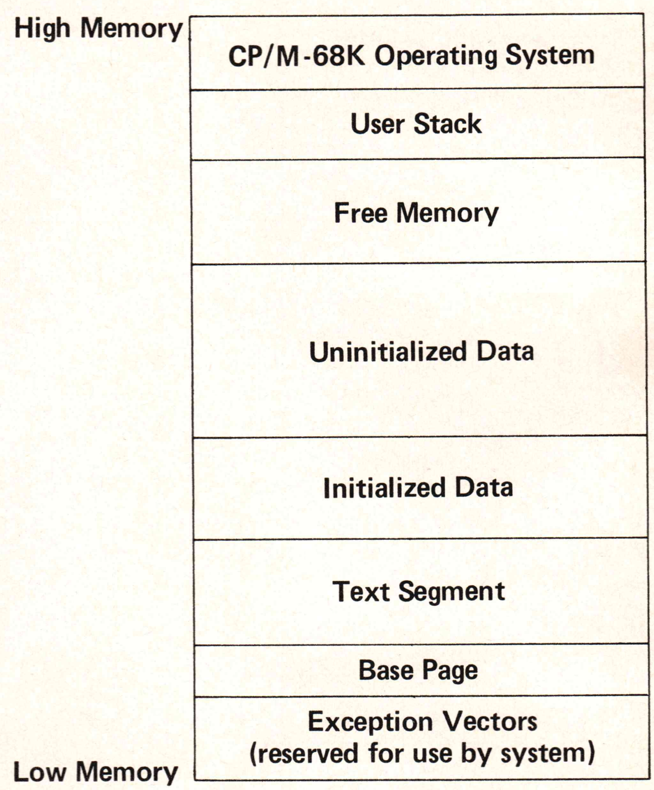


Figure 1.
CP/M-68K Memory Map

Any results are again returned in register D0.W.

Funtion 59 loads an executable program file into memory. The address of a Load Parameter Block (LPB) is passed in register D1.L. The LPB describes the program and specifies the load address; it includes the following items:

- address of File Control Block of successfully opened program file
- lowest address of area in which to load program
- highest address+1 of area in which to load program
- address of base page (returned by BDOS)
- default user stack pointer (returned by BDOS)
- loader control flags

The BDOS allocates memory for the desired program and base page and initializes locations 0000-0024H of the latter. Locations 0025-0037H are not initialized. The calling program must fix up any additional program, first pushing a return address on the stack if it wishes to resume execution later.

Funtion 61 (Set Execution) allows a program to link its own handlers to the operating system for such events as bus errors, illegal instructions, and zero divide.

Function 62 sets the calling program into Supervisor State and swaps to the system stack. Presently this call is always successful, though in future versions (especially multi-user systems) the function may not be present or may require certain privilege attributes.

In forthcoming columns we will provide examples and benchmarks of programs run under CP/M-68K, and begin to publish some 68000 utility subroutine listings.

Improved Square Root Routines

Don Taylor of Corvallis, Oregon writes: "If you are going to do square roots, and you have assembly language access, don't use Newton's Method. It is a fine tool, and very general purpose, but too slow for something as simple as this. I don't remember where I first ran across the integer bit-by-bit method, but typically it is an order of magnitude faster.

"The idea behind the enclosed square root routine is very similar to the way that it is done by hand, except for the fact that the process is even simpler in base 2. In decimal you use pairs of digits, in binary you use pairs of bits. In decimal you must guess the next trial digit, in binary it can only be zero or one.

"If you breakpoint the routine at the bottom of the loop, you can see the partial result growing bit-by-bit. In addition, the remainder should not exceed the trial root. If the remainder exceeds half the root at the end of the routine, the root can be better approximated by increasing it by one; without doing this the root will always be less than or equal to the real square root."

Our thanks to Don for this interesting and elegant subroutine. A listing in Intel 8086 mnemonics accompanies this column on page 82.

DDJ

(Listing begins on page 82)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 245**

Integer Square Root

(Text begins on page 79)

```
                                  title    'Integer Square Root'
                                  pagewidth 79
                          ;
                          ;  Integer Square Root for 8086/88
                          ;  Contributed by Don Taylor
                          ;
                          ;  Call with:      AX = Argument
                          ;  Returns:        AX = Square root
                          ;                  Other registers preserved
                          ;
0000 53                   sqrt:   push    bx        ;save registers
0001 51                           push    cx
0002 52                           push    dx
0003 8BD0                         mov     dx,ax     ;argument into DX
0005 B90800                       mov     cx,8      ;number of iterations
0008 33DB                         xor     bx,bx     ;clear the remainder
000A 8BC3                         mov     ax,bx     ;clear trial value and
                                                    ;final result store
000C D1E3                 sqrt1:  shl     bx,1      ;double partial result
000E 43                           inc     bx        ;guess next bit is a 1
000F D1E2                         shl     dx,1      ;fetch 2 new bits
0011 D1D0                         rcl     ax,1      ;from argument
0013 D1E2                         shl     dx,1
0015 D1D0                         rcl     ax,1
0017 2BC3                         sub     ax,bx     ;do a trial subtraction
0019 7308           0023          jnc     sqrt2     ;guess was right,
                                                    ;append a 1 bit
001B 03C3                         add     ax,bx     ;guess wrong, put it
                                                    ;back
001D 4B                           dec     bx        ;and clean up for
                                                    ;next pass
001E E2EC           000C          loop    sqrt1
0020 E90300         0026          jmp     sqrt3     ;go scale result
0023 43                   sqrt2:  inc     bx        ;convert xxxx01 to
                                                    ;xxxx10, i.e. append
                                                    ;a 1 bit
0024 E2E6           000C          loop    sqrt1
0026 D1FB                 sqrt3:  sar     bx,1      ;divide by 2 to get
                                                    ;actual square root
0028 8BC3                         mov     ax,bx     ;return result in AX
002A 5A                           pop     dx        ;restore other registers
002B 59                           pop     cx
002C 5B                           pop     bx
002D C3                           ret
                          ;

END OF ASSEMBLY.  NUMBER OF ERRORS:   0.  USE FACTOR:  0%

A>
```

End Listing

SOFTWARE REVIEW

Unica and XM-80

by Michael Favitta

Product: Unica and XM-80
From: Knowlogy
P.O.Box 283 – E
Wilsonville, OR 97070
$195.00 ($95.00 Unica only)
Reviewed by Michael Favitta

Knowlogy advertises that its software package "brings a Unix-like environment" to any Z80-based computer system running CP/M version 2 or beyond. The Unix environment is simulated by using a set of utility programs that possess Unix-like attributes. If you purchase XM-80, you will also need Microsoft's MACRO-80 assembler and LINK-80 loader.

The package is divided into two independent parts, the executable modules called Unica and the language translator XM-80. The Unica are a group of file-related utility programs that perform functions such as delete, copy and search. They are written in the XM-80 language and the source for each is provided. The XM-80 translator converts the XM-80 source modules into standard Z80 assembly language. The Unica may be purchased without XM-80 for $95. If desired, XM-80 may then be added at a later date for the cost difference plus a $10 handling charge. You may also purchase new and enhanced Unica as they become available but Knowlogy does not indicate what the fee for this service will be.

Unica

The Unica part of the package consists of eighteen COM files, each of which performs one function and is referred to as a command. To create the Unix-like environment, all commands support redirection of standard I/O to any device or disk file, multiple commands linked together via "pipes," and filename conventions that support user numbers and extended wild card capabilities.

The commands provided are:

BC	Binary file comparator that displays differences between two files.
CAT	Concatenates one or more files.
CP	Copies one or more files.
DM	Prints a disk allocation map.
FID	Computes a 16-bit checksum for a file.
HC	Concatenates one or more files horizontally (by line).
LN	Creates an alias for a file via a directory link.
LS	Lists the directory contents.
MV	Renames (moves) one or more files.
RM	Deletes (removes) one or more files.
SC	Source file comparator that displays differences between two files.
SFA	Sets and changes file attributes.
SP	Spelling error detector.
SR	Pattern searching with wild cards.
SRT	Sorts lines of a file by text in specified columns.
TEE	A "pipe fitting" used to direct output to more than one command.
WC	Counts the characters, words, and lines in one or more files.
WX	Extracts words from one or more files.

Each command has a set of flags associated with it that are used to specify procession options. Flags consist of a minus sign followed by one or more letters. They may appear separately or in groups, any place in the command line. Two of the most useful flags are the N and V flags. The N flag prints the name of each file as it is processed. The V flag takes this a step further and allows you to verify that processing should be carried out on each file specified. This allows wild card filenames to be used freely, as you can prevent any file from being processed even though it matches the wild card. This feature can prevent some costly mistakes when using a command such as RM to delete files.

Many of the commands are more useful than the brief descriptions of them above may indicate. The commands CP, HC, and SRT are such commands and merit further discussion. The spelling error detector (SP) also needs some explanation as it was the only Unicum that performed poorly.

The copy command (CP) has all the capabilities of PIP except for the data conversion functions (such as converting the output to upper case). In addition, CP allows you to set or change file attributes and will not overlay a file that is marked as write-locked. It can copy a file to the same disk so you can make backup copies easily on a single-disk system. The command is designed for maximum disk efficiency as it uses all available memory for buffers and always reads as much as possible before writing. In my tests, I found that CP was about twenty percent faster than PIP when two or more files were being copied. When copying one file they were about the same speed. As with all file-related commands, extended wild card capabilities are present that can be controlled with the V flag.

The horizontal concatenate command (HC) does a lot more than just append lines together. If you specify different sets of flags, HC will expand tabs to spaces, compress spaces to tabs, add leading spaces, remove trailing spaces, print one or more files side by side starting at any column position, and append a string constant to each line of a file. It is a handy tool for reformatting text to fit various terminal screen widths and for converting files from other systems to your own space/tab conventions.

The data sorting utility (SRT) sorts the contents of a file into alphabetical order. The size of the file that can be sorted is determined by your system's memory size, as sort is done in memory using the Quicksort algorithm. A primary sort key field can be specified by using flags to set the column positions. In this case the rest of the line will be used as a secondary sort key.

The spelling error detector (SP) creates a list of all the words in a file that it does not find a match for in its dictionary of 23,688 words. Unfortunately, the dictionary does not include plurals or conjugated verbs and there is no facility to add new words. These limitations are documented and Knowlogy promises to upgrade this command in the future. In actual testing, a document that contained 773 words had 680 of them flagged as possible errors. Only 74 of the words flagged were actually incorrect (e.g., misspelled, acronyms). Not wanting to believe the results, I created a file that contained 35 common words and ran SP again. To my disappointment, it flagged 28 of the words as possible spelling errors. This was also the only Unicum that I found any software problems with. SP causes a system crash if the dictionary file is not present when it is executed. In its current form, the SP Unicum is not worth using.

The Unica package is more than just a set of independent utility programs. The commands may be linked together to form complex procedures. These connections are called pipes. Along with the ability to redirect the output of a command to any device or file, pipes create a flex-

ibility that makes the Unica package a worthwhile investment. The following examples demonstrate the command line syntax, the redirection of output, and the use of pipes.

```
CAT file1 file2 |
    SR "find this" -1 >b:file3.ext;2
```

This command string concatenates file1 and file2 in a temporary file that is passed to SR. SR writes the line number and line to file3.ext on drive b for each line that contains the string "find this." File3.ext is associated with user number 2.

```
WX -v *.doc | TEE con: | SRT -u |
    TEE lst: >file1
```

This command string extracts a list of all words used in all files that have a .doc extension on the currently logged drive. You will be prompted to verify that you really want to process each file. The extracted words are printed on the console and piped to the sort (SRT) command to sort the list of words and remove all duplicates. The sorted list is written to the printer and file1.

With the exception of SP, I found the Unica package to be well though out with respect to what commands were provided and how they interacted when connected via pipes. Each command worked exactly as documented. Not all the commands worked perfectly in every case but the cases that caused problems were clearly defined in a "bugs" section for each command. Besides SP causing a system crash when the dictionary was not present, the only problem I encountered had to do with the physical size of the package. I have single-density 8-inch disk drives that have a formatted capacity to 250 Kbytes. The Unica without the spelling dictionary (another 73 Kbytes) take up 222 Kbytes of storage. This leaves very little room on a single disk for other utilities and text or source files. Since all temporary files that are created during processing (via pipes) reside on the currently logged disk, the problem is compounded. This problem can be overcome by using more than one drive. You then must specify the drive for all the files and commands that do not reside on the currently logged drive. This procedure can be quite frustrating as there is a much greater chance for human error, especially when entering complicated command sequences. Dual-density disk drives significantly enhance the practical usability of this product.

XM-80

XM-80 is a programming language that stresses the design of programs into modules that can be used in more than one program. The analogy of an electrical engineer, selecting existing integrated circuits and wiring them together to perform a required function versus using all discrete components, is used to describe this approach. Knowlogy calls this methodology "software synthesis." In keeping with the analogy, each module is called a component and related components are grouped into families. Two families of components are supplied with XM-80. The 2800 family consists of components for the Z80 that are not hardware or software dependent. The 2801 family are components that require CP/M version 2 or beyond. All the Unica are built from the components in these two families, "wired" together with Z80 assembly language. The source code for the two component families is not provided. Knowlogy's software licensing agreement does allow you to freely distribute any software that you develop, even if it contains one or more of the components. The agreement limits the distribution of any programs derived from a Unicum to other license holders.

XM-80 is an acronym for an extension of Microsoft's 8080/Z80 assembler, MACRO-80. The XM-80 compiler translates XM-80 procedures and/or procedure calls into standard Z80 assembly language that can be assembled by MACRO-80. Any line that is not recognized as an XM-80 statement is passed through unchanged, allowing any mixture of assembly language and XM-80 to be used in writing XM-80 programs. Using XM-80 forces programmers to abide by a rigidly defined set of interface rules when writing new components. These include component naming and documentation conventions. By forcing adherence to the component interface rules for each component, the XM-80 compiler can validate that the arguments passed and the registers used by the specified procedure are correct. The end result of using a tool such as XM-80 is to increase programmer productivity and decrease program development time by not reinventing the wheel for each new application. The compiler also reduces the possibility of program errors caused by passing parameters to subroutines incorrectly.

The following is a summary of XM-80 programming guidelines and syntax.

The rules for XM-80 program development are:

(1) Search through the available components and select those which are applicable to the project.

(2) Define any new components needed. Code, test, and document each new component. Enter the completed component and documentation into the correct family library.

(3) "Wire" together the components with assembly language to produce the completed program.

The interface rules for components are:

(1) Component names consist of up to six upper-case letters to identify the component, followed by a four-digit family name, followd by two lower-case letters to denote the author. If more than one global symbol exists for a component, then three letters are used to identify the component and global symbols are created by appending up to three letters to the component name.

(2) The syntax for procedure definition and procedure calling is demonstrated by the examples below.

```
proc GETNUM [CHAN:stk] →
    [ERROR:a,NUM:hl] +C-de
begin
Source code (can include calls to
    other components)
end GETNUM
```

The procedure GETNUM is defined as having one input parameter that is passed on the stack. An error code is returned in the accumulator and a number in the HL register pair. The carry flag is altered and register pair DE is destroyed.

The GETNUM procedure is called by:

```
GETNUM [stk=(VALUE)] →
    [a,SAVEHL=hl] +C-de
```

The contents of the 16-bit word pointed to by VALUE are pushed onto the stack and then the procedure GETNUM is called. The contents of register pair HL are stored at SAVEHL after the return.

The procedure definition is also entered in a special file called the component interface file (CIF). The CIF is used for parameter error checking by the XM-80 compiler. Note that the registers destroyed by the called routine must be listed on the procedure call. These are checked by the compiler to make sure that the programmer is aware of all registers that will be affected by the procedure call.

The XM-80 compiler understands several statements besides those used in procedure calling and definition. These

PLACE
STAMP
HERE
The Post Office
will not deliver
mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems

EDITORIAL DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94025

Dr. Dobb's Journal
For Users of Small Computer Systems

Reader Service Card

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date.

Name ______________ Address ______________

City ______________ State __________ Zip __________ April 1983 #78

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
Articles:		227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

Profession:

1☐Programmer/Analyst
2☐Engineer/Scientist/Technician
3☐Business Owner/Manager
4☐Educator
5☐Consultant
6☐Professional (Law, Medicine, Accounting, etc.)
7☐Student
8☐Other ______________

Brand name of microcomputer you work with:

1. ______________
2. ______________
3. ______________

Purchase of magazine:

1☐Subscription
2☐Computer Store
3☐Newsstand
4☐Bookstore
5☐Passed on by friend/colleague
6☐Other ______________

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal
For Users of Small Computer Systems

P.O. BOX E
MENLO PARK, CA 94025

Dr. Dobb's Journal
For Users of Small Computer Systems

Reader Service Card

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date.

Name ______________ Address ______________

City ______________ State __________ Zip __________ April 1983 #78

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
Articles:		227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

Profession:

1☐Programmer/Analyst
2☐Engineer/Scientist/Technician
3☐Business Owner/Manager
4☐Educator
5☐Consultant
6☐Professional (Law, Medicine, Accounting, etc.)
7☐Student
8☐Other ______________

Brand name of microcomputer you work with:

1. ______________
2. ______________
3. ______________

Purchase of magazine:

1☐Subscription
2☐Computer Store
3☐Newsstand
4☐Bookstore
5☐Passed on by friend/colleague
6☐Other ______________

Comments:

PLACE STAMP HERE
The Post Office will not deliver mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems
ADVERTISING DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94025

PLACE STAMP HERE
The Post Office will not deliver mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems
ADVERTISING DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94025

Dear Reader,

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps, or for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take the time to write a letter. This card provides you with a quick and easy way to correspond. Simply fill it out and drop in it the mail. We take care of the rest.

Thanks for taking a few minutes to talk with us.

– Editor

Which articles or departments did you enjoy the most this month ? Why ?

____

____

Comments or suggestions:

____

____

____

____

statements are used to give the programmer control over how XM-80 sets up procedure linkages and where needed CIF files can be found. The compiler can also be instructed to generate inline code if desired. The compiler itself is not a Unicum so it does not support any of the Unica capabilities such as redirection of output. It uses a command syntax similar to that of MACRO-80. Most of the command line is used to generate a MACRO-80 call after XM-80 is done. The XM-80 compiler is controlled by specifying a set of flags similar to those used with the Unica at the end of the command line.

Again, the package worked as documented although the "known bugs" section documented some serious problems (note that the version I reviewed was released in March 1981). The worst of these is that if you enter the extension on the source input file, it is deleted. I would have liked to have seen a warning at the very front of the manual about this problem, as it is an easy mistake to make and the consequence could be hours of lost work. The XM-80 package is a worthwhile assembly language programming development tool. It includes an extensive set of library components that could save many times the price of the package in programmer time. An added bonus is that by using the I/O components provided, your programs can contain all the features that the Unica possess, including the ability to build pipes.

Unica and XM-80 Documentation

The documentation for Unica and XM-80 comes in a three-ring notebook with labeled dividers that make it simple to find a desired section. This format makes it easy to add to and update the manual as you develop new Unica or XM-80 components. My one big gripe with the manual is that it doesn't have a usable page-numbering scheme. All Unica and components start at page 1. This makes an index useless, so Knowlogy did not include one. There isn't a table of contents or any kind of summary of which Unica and components come with the package either. This means you have to read the entire manual to get a feel for what the package offers. The manual becomes tedious to use once you've become familiar with the package and just want to use the manual as a reference. This problem could be solved easily by the addition of a couple of appendices with summaries of the Unica and XM-80 components and their associated syntax and argument lists.

Putting format aside, the manual is written carefully and thoroughly describes the product and how to use it. Examples are used liberally throughout the text to demonstrate what the various Unica do and how they can be combined to perform other functions. For XM-80, an example is provided that takes you from program design, through testing, to program completion. Detailed data sheets are provided for each of the components. The source codes for the Unica can also be used as an excellent tutorial on how to write XM-80 programs, as they are well written and extensively commented.

Knowlogy dispenses a good deal of software design philosophy along with the hows and whys of Unica and XM-80. This makes the manual read more like a textbook than a reference manual. (This probably has something to do with why they chose "Knowlogy" as a company name.) The entire package is presented as a complete methodology on how to solve problems with a computer, with the Unica part of the package an end result of the methodology. Personally, I like this approach and think the resulting manual is one of the best that I have seen with respect to content.

Summary

The Unica are a group of utility programs that use an extended CP/M command sequence that allows the output of one command to be used as the input to another. With the exception of the spelling command (SP), all the Unica worked as documented and would make a useful addition to just about anyone's set of utility programs. Even without SP, I think the Unica package is well worth $95.

XM-80 is a software development tool designed to increase programmer productivity. If the "software synthesis" methodology is followed, then this tool is a worthwhile investment. If you can't live within the rigid framework defined by the methodology, then I would suggest saving the extra $100 that XM-80 costs. To get the most out of the entire package, you should design your programs with the existing components that provide the special features used by the Unica.

Overall, the Unica/XM-80 software product is designed and documented very well. The package I received did have a few problems, which I presume will be corrected in later releases, but most are documented and can be programmed around.

We checked with Knowlogy to see if there had been upgrades from the release that our reviewer received. We were informed that they have indeed released an upgraded version that has several improvements, among them a redone spelling error detector which now has a dictionary of 130,000 words (including plurals and conjugations) and the ability to add new words. – Ed.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 247**

OF INTEREST

by Michael Wiesenberg

Baudy Modem

Modems with 1200 baud are lowering in price. **Racal-Vadic's VA212LC** costs $550. The Bell-compatible, switched-network, originate/answer full-duplex modem operates at 1200, or 0 to 300 for communications with slower systems, automatically determining the baud rate of a calling modem. Auto-answer can be disabled to obviate answering calls on lines that carry both voice and data. Five LEDs display operational status, monitoring transmit and receive data, carrier detect, voice/data status, and bit rate; and the unit continually monitors itself with automatic self-test routines. The VA212LC automatically handles 9- or 10-bit character codes, and thus transmits EBCDIC or ASCII. **Reader Service No. 101.**

A Genius for Displays

Many of the newer dedicated word processing systems and computers have full-page screens, but now you can get an 80-character x 57-line display (73 optional) for Apple and most other computers with standard RS232 port. **The Genius** from **Micro Display Systems, Inc.**, offers white, green, or amber phosphor, reverse and flashing video, 128-character ASCII set (or optional foreign character sets), graphics, 19.2K baud interface, and internal 16K memory buffering and screen memory, on an 8- x 10.5-inch display. You can get 100/120 volts, 60 Hz for $1795, or shielded 220/240 volts, 50 Hz for $1950. (Prices are FOB Minneapolis, which, I think, means you have to pay for shipping from there to anywhere else.) **Reader Service No. 103.**

And Another Big Monitor

QuadScreen, from **Quadram**, for the IBM PC is a 17-inch monochromatic display with 160 characters x 64 lines, more than five times the screen capacity of IBM's monitor. Bit-mapped graphics permit individual addressing of all 960 x 512 dots, while the hardware can display any character font in any size (providing infinite user definable character sets). QuadScreen has its own character set in a 5 x 7 matrix, while software provided by Quadram permits defining characters of any size or shape. Reverse video and smooth forward and backward scroll are part of the product. $1950 gets you the monitor, cable, software, and display controller board that fits into any expansion slot of the PC and has 64K memory that can be used alone for memory expansion.

Quadram also makes **QuadSpooler**, a software program that permits simultaneous printing and spooling, even with two printers. You designate from 2K to 56K of RAM to use as the spooler buffer. No additional hardware is required, though Quadram would like to sell you one of their Quadboard memory expansion boards for the PC. Buy one and get QuadSpooler free, or get just the software for $19.95.

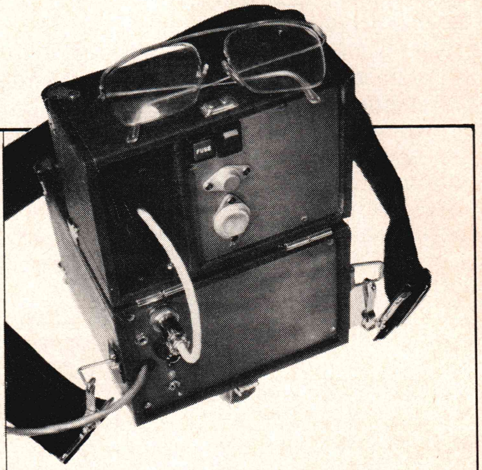
And while I'm giving them space, Quadram's **MicroFazer** has improved since last I mentioned it. This hardware buffer/spooler has a copy feature that prints as many additional copies of the buffered information as specified by pressing a button. Use it with any computer or printer, and expand its 8K memory size up to 512K (without using any of the computer's memory). Parallel-to-parallel, parallel-to-serial, serial-to-parallel, and serial-to-serial are all available, priced from $169 to $1395. **Reader Service No. 105.**

And An Inexpensive Amber Guy

The **ADP-120A** from **Atlantic Data Products**, a 12-inch, high-resolution, amber phosphor (amber on grey), 16MHz-bandwidth data monitor, compatible with many micros, costs $169. **Reader Service No. 107.**

Rufus T?

The **Firefly** battery pack (and eyeglass stand) from **Gamma Research** gives three to eight hours of portable battery power to Osborne, Apple, IBM, KayPro, Franklin Ace, et al. An optional uninterruptible power supply (U.P.S.) protects micros from voltage spikes or blackouts by automatically switching to the Firefly whenever sensing AC failure and functions as a rapid (three- to four-hour) Firefly



charger. You get a 12-volt DC power cable to run the computer or charge the Firefly from your car's cigarette lighter, four LED charge state monitors, a discharge warning buzzer, and a 12-month warranty. It all comes in a detachable (into two sections) 6½- x 4½- x 9½-inch hard travel case with a carrying strap. The Firefly battery pack reduces computer heat generation by up to 80%, they say, which extends the computer's life. The battery pack is $229 to $279, and U.P.S. $99 to $109. And to double battery power time, they have a booster pack for $169. Gamma Research also offers a Firefly 9-inch, 1000-line, high-resolution, green ($244), or amber ($299) monitor in a lightweight travel case with a detachable daylight glare-reducing hood, that runs off AC, your car's cigarette lighter, or (or course) the Firefly battery pack. **Reader Service No. 109.**

Basic Database System for CBASIC and Other Bases

The **Tarbell Database System** from **Elliam Associates** is supplied in .COM files *and* CB80 source (to run under CBASIC). It supports up to 19 open files, with no limit on numbers of records or their lengths or those of field names. The system's series of programs uses a common file format for random and sequential files, for optional index files, and to chain from the main menu to other programs and HELP files, set up files, enter data, update files, write reports, build command files, copy files from one format structure to another, sort files, print mailing labels, and personalize letters. A QUERY language interactively or from command files defines search area, scope of search, and search conditions. You need a 48K sys-

tem. $100 for disk in most configurations and formats (specify), plus $1.50 p & h (Californians, add sales tax). **Reader Service No. 115.**

Yeah, But Where's Ferddy?

MORTTY is a communications program from **Phillip Emerson** for H8, H89, Z89, and Z90 computers. It sends and receives ASCII and Baudot code at any baud rate the equipment can handle, uses a full screen and a user-definable split screen, has a type-ahead buffer, logs on automatically, allows the user to define handling of carriage return, line feed, and control characters, echoes or not as defined, uses full or half duplex, automatically identifies Morse code, has advanced string storage, automatically programs smart modems, printers, etc., and has user-definable, single-keystroke functions. You need HDOS with 32K and a modem. You get a 5¼-inch hard-sectored disk, a 60-page manual, and a year's free updates, for $100 (Ohio, add sales tax). **Reader Service No. 117.**

Alien Contest

The **Alien Group**, who makes Voice Box speech synthesizers for Atari and Apple computers, would like to give away $6,800 to the authors of what they judge to be the best talking or singing programs that use (what else?) the Voice Box, in their **Voice Box-ing Match Contest** (cute, cute). First prize is $5,000. Judges will be computer game players ages 13 to 18. The Alien Group also wants you to know about their new Atari Voice Box that recognizes phonetic or normal spelling, has an (apparently) unlimited vocabulary and random sentence generator, is programmable from BASIC or assembly language, plugs directly into the serial port, does not blank the screen while talking, is available in 16K cassette and 16K and 32K disk versions, speaks with intonation or feeling (but they don't say if it can do *both* simultaneously), sings in key (that's better than I can do), sings in three simultaneous musical tones, converts two keyboard rows into a literal chromatic piano with a 2½-octave range, programmatically handles glissando, tremolo, and vibrato, comes with 40 songs that you can add to, has a system for easily (so they say) creating and editing words and music, has educational software, has two "talking heads" with programmable lip-sync animation and high-res color, and inclues a demo. $169 for a wonderful product that the *Las Vegas Sun* thought was the highlight of COMDEX. **Reader Service No. 123.**

Process Words for Less

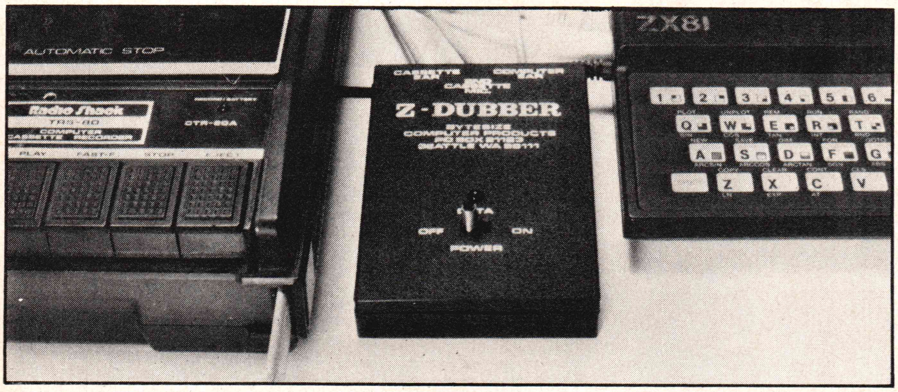
Word processing software can cost upwards of $500 and may not be worth the price, or cost under $100 and not work worth beans. Here's one from a company known for quality software (in the games field mainly, but they do have a reputation to maintain) that seems to offer a lot for $69.95. **Bank Street Writer** from **Brøderbund Software** ("Developed," they say, "and heavily tested among students and young adults by Bank Street College and Intentional Educations" – is that the name of a *real* school?), for 48K Apple II+ or Apple II with Applesoft in ROM or RAM and 16-sector controller and for 48K Atari 400/800 (and maybe 1200?) (requiring BASIC cartridge for tutorial only), has global search and replace, move and unmove, automatic centering and indent, inverse text highlighting, word wrap, disk storage and retrieval functions with password protection (you don't get *that* on most WPs!), redefinable defaults, and a "potent" print format routine, including document chaining, page headers, page numbering at top or bottom, partial printing, and format previewing before printing. No special hardware is needed and upper and lower case are displayed. The back of the disk has a tutorial, and you get a free backup disk and complete documentation (I think that means a manual). **Reader Service No. 119.**

"-trix" Are Not Necessarily for Kids

Quotrix, the second of **Insoft**'s "-trix" educational games for the IBM PC has players guess one of 700 possible famous quotes by following a trail of crossword puzzle clues, trivia questions, foreign words, and other word games. Sounds like a lot of fun for $34.95. **Reader Service No. 121.**

ET Enthusiasts

Join **ETUG**, the user's group for Heath's ET/ETA-3400 microprocessor trainer (but not affiliated with Heath) and for those who program the 6800 and 6809 and in tiny BASIC, and receive a quarterly newsletter for $16 a year (US and Canada; $22 elsewhere). They've been around for over a year. **LAETUG** for Los Angelenos has just fired up, and meets at a Heathkit Center. ETUG, 11231 Oak St., El Monte, CA 91731; LAETUG, c/o Gilbert



Load Bettair Weeth Z-Dubber

Improve on the current difficulty of loading cassette programs into the Sinclair ZX81/Timex 1000 with **Z-Dubber** from **Bytesize Computer Products** or connect two cassette recorders for making perfect backups. $29.95 plus 90¢ shipping. **Reader Service No. 111.**

Murillo, Heathkit Electronic Center, 2309 S. Flower, Los Angeles, CA 90007; (213) 749-0261 and 443-2237.

Tick TOC

The **Technology Opportunity Conference** (TOC), a worldwide convergence of optical storage, videodisk, and computer technology sponsored by *OPTICAL MEMORY NEWSLETTER Including Interactive Videodisks* and Office of the Future Limited (heady names indeed!), will meet in San Francisco, April 5-7; New York, April 26-28; Washington, June 14-16; London, July 5-7; Los Angeles, July 11-15; New York, September 12-16; Chicago, October 10-14; San Francisco, November 8-10; Kaanapali (Maui), November 29-December 1; Orlando, December 5-9; and Houston, January 9-13. **Reader Service No. 127.**

Logo For Atari

Logo for the Atari 400, 800, and 1200XL, developed and manufactured for Atari by Logo Computer Systems, Inc., of Montreal, will soon be available in a 16K cartridge sold and distributed by **Atari** for under $100. **Reader Service No. 113.**

Learn Modula-2

From the Father of Pascal, Nicklaus Wirth, came Modula-2 (*MODU*lar *LA*nguage) in 1980, sometimes called "Pascal for grownups," and described in understated fashion by Wirth himself this way: "The structure of Modula is an improvement upon the structure of Pascal." Yes, there was a Modula-1, the developmental version, in 1975. **Logitech** is developing Modula-2, together with a fully symbolic debugger, for various 8086 CP/M-86 and MP/M-86 environments, and would be pleased if you would contact them for more information. **Reader Service No. 129.**

And to learn all about the language, **Volition Systems** are selling a 264-page loose-leaf format **Modula-2 User's Manual**, accompanied by Wirth's 48-page Modula-2 technical report, for $35. It contains information about standard library modules, utility library, and implementation on UCSD Pascal, and a machine-specific implementation guide with information on libraries, interrupt handling, and machine-level data representation. You can also buy Wirth's Springer-Verlag book, **Programming in Modula-2**, from them for $16. **Reader Service No. 131.**

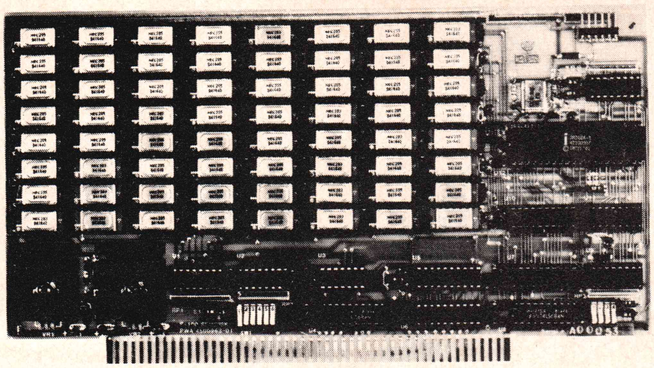
Can Aerobics for the Computer Professional Be Far Behind?

I've got my copy of **Tone Up at the Terminals**, "an exercise guide for workers in automated offices." You can get yours too, *free*, from **Verbatim Corporation**. Now, without leaving the confines of my terminal, I can trim my hips and waistline by doing the Windmill, release tension in my hand and wrist with the Wrist Flex, do the Knee Kiss, and, to firm and tone my legs and (blush) buttocks, I can do the Derriere Firmer. The 12-page booklet includes over 20 exercises that have been reviewed and approved by the California Governor's Council on Wellness and Physical Fitness. **Reader Service No. 133.**

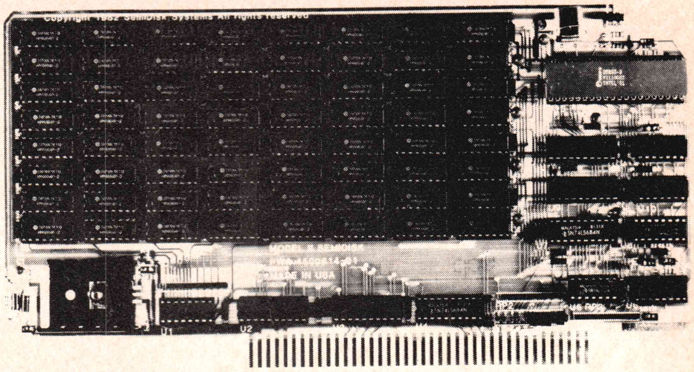
DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 249**

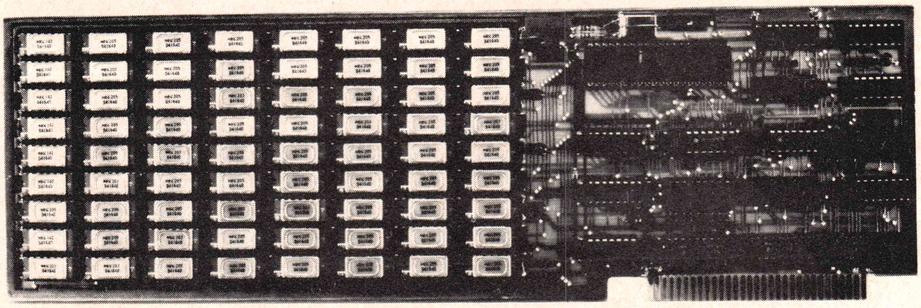
A FULL LINE OF SEMIDISKS



S-100



TRS 80 Model 2



IBM Personal Computer

Do you use your computer? Or does your computer "use" you? Face it, if you're using floppies, your time is being wasted. Because a floppy is an inefficient random access storage device. Each time the processor wants to transfer data, it has to wait an eternity for the disk to rotate and the head to move.

So what do you do? Get a SemiDisk, quick. It's a large capacity semiconductor memory board that is driven by software to operate like a disk drive. Without all the waiting. Do everything you'd do on a floppy or hard disk, with no modifications to your software or hardware. Two board sizes are available: 512K and 1 Megabyte. (the highest density microcomputer memory board in the world) And you can put up to 8 megabytes in a system by adding more storage boards.

What do you need to use it? Just an S-100 system with CP/M 2.2. Or a TRS-80 Model 2 system with CP/M 2.2. Or an IBM Personal Computer. That's it. No special processors, DMA, I/O, or disk controllers are required. Plug it in and run the installation program, and you're on your way. Fast! Even better, we supply full source code to the driver software, in case you'd like to do your own interfacing.

Best of all, the SemiDisk's price won't warp your wallet. Compare specs, cost/megabyte, storage capacity, and compatibility with the competition. You'll see that the SemiDisk is a disk emulator truly worthy of the name. SemiDisk has battery-backup capability, too.

Consider our limited warranty: A full year, covering all parts and labor. Consider our liberal 15 day return policy. Price? $1995 for 512K byte SemiDisk, $2995 for 1 Megabyte SemiDisk. Both from stock. $10.00 for manual. VISA, Mastercard, COD orders accepted. Dealer and OEM inquiries welcomed. (Specify system type and disk format when ordering.)

Someday, you'll get a SemiDisk.
Until then, you'll just have to . . . . wait.

SemiDisk SYSTEMS
P.O. Box GG
Beaverton, OR 97075
(503)-642-3100

Call (503)-646-5510 for CBBS®/NW, a Semi-Disk-equipped computer bulletin board.
SemiDisk trademark of SemiDisk Systems; TRS-80 trademark of Radio Shack

Circle **no. 33** on reader service card.

INTERSTELLAR DRIVE™

A SOLID STATE DISK EMULATOR

Save valuable time!
5 to 50 times faster performance than floppy disks and Winchester drives

PION'S INTERSTELLAR DRIVE is designed for use with a family of interfaces and software packages. Currently available are interfaces for IBM, S100, TRS80, Apple, SS50, and most Z80 uP, and software for most popular operating systems. Additional interfaces are continually being developed for the most popular computers.

SAVE MONEY!
Increase your computer's productivity

The INTERSTELLAR DRIVE is a high performance data storage subsystem with independent power supply, battery backup, and error detection. It has 256KB to 1 Megabyte of solid state memory integrated to perform with your operating system.

Basic Price for 256KB unit [includes interface and software]
$1095. *plus tax (where applicable) and shipping*
Visa and Master Card accepted. MasterCard VISA

PION, INC. Tel.(617)923-8009
101R Walnut St., Watertown, MA 02172

TRS80 trademark of Tandy Corp. Apple trademark of Apple Computers
Interstellar Drive trademark of PION, Inc.

Circle **no. 41** on reader service card.

CP/M® for the 68000

Fully configured hard disk development systems, based on CompuPro hardware, are now available from Gifford Computer Systems. The total price with hardware and software? Only $9990. All you need is an RS-232 terminal to be up and running. Each system is covered by a two year, 24 hour replacement warranty. Systems are available for delivery now.

Hardware

- ☐ 8 MHz Motorola MC68000™ CPU
- ☐ 256K high speed static RAM
- ☐ 2.4 Mbyte floppy disk storage
- ☐ 21 Mbyte formatted Winchester disk
- ☐ 9 serial ports
- ☐ Real time clock/calendar
- ☐ 20 slot IEEE 696/S-100 motherboard

Software

- ☐ CP/M-68K™ from Digital Research
- ☐ C compiler from Digital Research
- ☐ 68000 assembler from Digital Research
- ☐ Mince visual editor from Mark of the Unicorn

Upgrades for CompuPro based computers.

- ☐ CompuPro™ 68000 CPU (8 MHz) . . . . $ 850.00
- ☐ CP/M-68K™ from Digital Research . . . . $ 350.00
- ☐ 20 Mbyte Winchester disk subsystem with CBIOS drivers $3,595.00

GIFFORD COMPUTER SYSTEMS
1922 Republic Avenue, San Leandro, CA 94577
(415) 895-0798 A division of G&G Engineering
I'D LIKE THE WHOLE STORY.
Please send me your brochure.
Name______ Title______
Organization______
Address______
City______ State______ Zip______
Phone______
☐ Please have a representative call me.

CP/M is a registered trademark of Digital Research. CP/M-68K is a trademark of Digital Research. MC68000 is a trademark of Motorola. Mince is a trademark of Mark of the Unicorn. CompuPro is a trademark of Godbout Electronics.

GIFFORD COMPUTER SYSTEMS CENTERS ☐ SAN LEANDRO, CA 94577 (415) 895-0798 ☐ SAN FRANCISCO, CA 94104 (415) 391-4570 ☐ LOS ANGELES, CA 90064 (213) 477-3921 ☐ OKLAHOMA CITY, OK 73112 (405) 840-1175 ☐ HOUSTON, TX 77046 (713) 877-1212 ☐

Circle **no. 72** on reader service card.

Now for Concurrent CP/M-86

VEDIT - The Clear Choice for Programmers

Plus Features for Fast & Efficient Word Processing

Increasing your productivity is what a good text editor is all about. VEDIT excels by giving you a unique combination of extensive and easy to use editing features, customizability and complete hardware support. So compare VEDIT. You'll find everything you expect in a good editor plus a variety of time saving features which only VEDIT offers.

VEDIT is fully user oriented. You can use the function keys on any keyboard, or a layout you are already familiar with - simplifying your usage and easing your learning. While most editors lose text if you run out of disk space, VEDIT lets you delete files or change disks. VEDIT is the result of continuous enhancement and feedback from our nearly 4000 users.

VEDIT
CP/M
CUSTOMIZABLE
FULL SCREEN EDITOR

For program development it surpasses any other editor - with more extensive file handling, important command macro capability and special features for Pascal, PL/1, 'C', Cobol, Assembler and others. With VEDIT you will reduce your program editing time by 30% as compared to the best word processor.

For word processing, VEDIT has word wrap, adjustable margins, reformatting of paragraphs, word and paragraph functions and simple printing with imbedded printer control characters.

Command macros let you perform editing tasks you might otherwise not even attempt. Time consuming tasks for other editors (such as translations or extensive search/replace on many files), can be done by VEDIT without your intervention, even overnight if you choose.

VEDIT supports all of the new CRT terminals, video boards and 8080, Z80 and 8086 computers. We have been consistently first to support new computers - first for CP/M-86, first for MSDOS. And we will support you with any technical assistance you may need.

For the full story, purchase VEDIT risk free. Evaluate the 125 page manual and if you are not satisfied, return the package (disk unopened) for a courteous refund.

COMPARE VEDIT

Feature	VEDIT	WordMaster	WordStar
True Full Screen Editing	Yes	Yes	Yes
Edit files one disk in length	Yes	Yes	Yes
Compact and fast	Yes	Yes	No
Display of line and column #	Yes	No	Yes
Set/Goto text markers	Yes	No	Yes
'Undo' key to restore line	Yes	No	No
Automatic Indent/Undent	Yes	No	No
Adjustable tab positions	Yes	No	Yes
Repeat function key	Yes	Yes	No
Text move and copy	Yes	Yes	Yes
Scratchpad buffers	10	Only 1	No
Load/Save buffers on disk	Yes	No	No
Flexible command mode	Yes	Yes	No
Multiple command macros	Yes	No	No
Directory display	Yes	No	Yes
Edit additional (small) files simultaneously	Yes	No	No
Insert another disk file	Yes	Yes	Yes
Unlimited file handling	Yes	No	No
Automatic disk buffering	Yes	Yes	Yes
Recovery from 'Full Disk'	Yes	No	Some
Change disks while editing	Yes	No	No
Startup command file	Yes	No	No
Program CRT function keys	Yes	No	No
Word Wrap and reformatting	Yes	No	Yes
Printing	Simple	No	Extensive
Print formatting	No	No	Yes
Menu driven installation	Yes	No	Yes
Support newest CRT terminals	Yes	No	No
Support smart CRT functions	Yes	No	Some
Customizable keyboard layout	Yes	No	No
Available for CP/M-86	Since 1981	?	?
Available for MSDOS	Since 1981	?	Yes

Please specify your microcomputer, video board or the CRT terminal version, 8080, Z80, or 8086 code, operating system and disk format.

VISA & MasterCard

VEDIT - Disk and Manual
For 8080, Z80 or IBM PC $150
For CP/M-86 or MSDOS $195
Manual only $18

Zenith Z100 and Z89 • DEC VT180 • Televideo 802
TRS-80 I, II and 16 • Xerox 820 • Apple II Softcard
SuperBrain • NorthStar • Cromemco • Altos • Vector
MP/M • CP/M-86 • MP/M-86 • MSDOS • PCDOS

IBM Personal Computer and IBM Displaywriter

CompuView
PRODUCTS, INC.

1955 Pauline Blvd., Suite 200 • Ann Arbor, Michigan 48103 • (313) 996-1299

IN AUSTRALIA DISTRIBUTED BY SOFTWARE SOURCE PTY. LTD.
89 OXFORD ST., BONDI JUNCTION, SYDNEY - (02) 389-6388

Circle no. 86 on reader service card.

ADVERTISERS INDEX

EXTRA

EXTRA

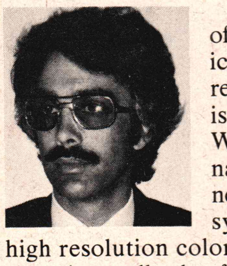
S-100 World News

MACROTECH International Corporation — 22133 Cohasset Street, Canoga Park, California 91303 • 213-887-5737

Image achieved by DGS' CAT 1600 Series color video graphic workstation. Picture courtesy of Digital Graphic Systems, Inc. See story below.

GRAPHICS: NOW MAX-IMIZED

CANOGA PARK—March 30, 1983—The decreasing costs and increasing density of memory made possible the present boom in digital graphics. Graphic systems designers are now able to take another major step with the introduction of MAX-M, a one megabyte memory board for $1983. As large size system memory and multi-megabyte Virtual Disk, MAX-M opens up major new low cost implementations.



Wayne Maw, Director of R&D for RGB Dynamics, Salt Lake City, Utah, reports, "My application is dependent on speed. With the Macrotech dynamic board, I have the needed speed." The RGB system is a Z80-based, high resolution color directory system for shopping malls, due for April release.

Empirical Research Group of Kent, Washington, creates a state-of-the-art high resolution color video graphics system by integrating their fast 68000 computer, Macrotech system memory, and the color video image processor from Digital Graphic Systems, Inc., Palo Alto, California. Radcliffe Goddard of Digital Graphics states, "High speed image processing requires large system memory to provide instantaneous display frame paging."

The demand for MAX-M by the graphics industry was nearly instantaneous following the initial Macrotech announcement.

MAX—256K to 1M S-100 Memory

CANOGA PARK—March 30, 1983—Mike Pelkey, Macrotech International president, today released details of the revolutionary MAX line of S-100 memory boards. Pelkey stated: "IEEE-696 now has a new standard for dynamic memory. The MAX product line offers 256K to 1M, at a price that ranges down to less than $0.00023 per bit." Pelkey continued, "The MI product line now includes our ultra fast (70 ns) 128K static memory, with battery backup capability, plus the 150 ns dynamic memories—in every 128K step from 256K through 1M (1024K) bytes, and add-on kits to permit field upgrade of sizes."

The extreme density of the MAX family is made possible through the use of proprietary PALs (programmable array logic). Also stated as available for add-on to any size MAX is Macrotech's popular M³ memory mapping architecture. M³ permits the 16-bit address space of an 8-bit processor to be dynamically mapped in 4K pages into as much as 16 megabytes of physical memory.

Parity error detection and 8/16 bit data transfer capabilities are provided as standard on the MAX series memory board.

PRICE INDEX

	SIZE	P/N	PRICE
Static Memory	128K	128-ST	$1232
Dynamic Memory 24-bit Addressing	256K	MAX-256	$1108
	384K	MAX-384	1292
	512K	MAX-512	1647
	768K	MAX-768	1815
	896K	MAX-896	1859
	1M	MAX-M	1983
With 16-bit M³ Addressing option, add $91			
	FROM/TO	**P/N**	**PRICE**
Upgrade Kits	256K/384K	MKT-2/3	$ 192
	256K/512K	MKT-2/5	692
	256K/768K	MKT-2/7	876
	256K/896K	MKT-2/8	967
	256K/1M	MKT-2/M	1060
	384K/512K	MKT-3/5	600
	384K/768K	MKT-3/7	784
	384K/896K	MKT-3/8	876
	384K/1M	MKT-3/M	968
	512K/768K	MKT-5/7	284
	512K/896K	MKT-5/8	376
	512K/1M	MKT-5/M	468
	768K/896K	MKT-7/8	192
	768K/1M	MKT-7/M	284
	896K/1M	MKT-8/M	192
M³ option		MKT-M3	121
Software (provided on 8" disk) Virtual Disk for MP/M II* and CP/M 2.2, CP/M 3.0* Bios modules, CP/M memory tests			$ 25
Manuals (sold separately) 128/ST			$ 15
MAX Technical Manual			15

Software for M³ Available

BURBANK—March 30, 1983—"M³ bank switching for 8-bit processors is much more useful with the new creative systems programs," states Dan West of Westcom Systems Inc. MP/M II* disk intensive applications are greatly improved with the new Virtual Disk routines now available through Macrotech OEM's and dealers for their M³ memory boards.

Westcom Systems, as the software consulting firm for Macrotech, has also provided subroutine listings to easily incorporate M³ mapping into the new CP/M 3.0* (CP/M Plus*) Bios module. The advantages of CP/M 3.0* with disk buffering, hashed directories, and user program expansion go hand in hand with Macrotech's flexible "bank switched" memory capabilities.

All Macrotech software and manuals are available through Dan West's Compuserve account #70250,102.
Leave comments/questions as E-Mail.

These new techniques can combine the above features with custom needs of the future, such as printer buffering, multi-page display and memory-intensive graphics displays.

The software listings are included in the Macrotech memory board manuals and are optionally available on 8" diskettes.